Declaration

Word Count Of Thesis: 38,878

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Candidate Name: Maximilian John Walker

Signature: m.walker

Date: 23/07/2018

This thesis is the result of my own investigations, except where otherwise stated. Where correction services have been used, the extent and nature of the correction is clearly marked in a footnote(s).

Signature: m.walker
Date: 23/07/2018

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signature: m.walker

Date: 23/07/2018

Maximilian John Walker
2018

Capture, classification and path-planning of safe routes using an aerial camera platform

Maximilian John Walker
Department of Computer Science
Aberystwyth University

This dissertation is submitted for the degree of
Doctor of Philosophy

Aberystwyth University 2018

In loving memory of Robert Henry Wallace Walker, who unfortunately passed away during the course of this project. Without his help and support it would not have been possible.

Acknowledgements

I would like to thank my supervisors Dr Hannah Dee and Dr Mark Neal for all of their help, particularly during the most challenging periods of this work and my fiancée Jennifer Smith for putting up with me during these times. In addition I'd like to thank Ian Izett and Peter Todd for their technical assistance.

Abstract

In real world situations it is sometimes impractical for a human being to enter or traverse an area due to safety or feasibility constraints. In these areas, robotic vehicles are commonly used to navigate and achieve tasks in place of a human subject. Unfortunately ground and waterborne vehicles suffer from a limited world view which may limit their ability to plan safe or optimal routes. By working in tandem with an aerial vehicle, these ground based robots can increase their world view with a resultant improvement in safety, speed or optimality.

We have developed a closed loop system for the navigation of unknown environments using a two vehicle team. A Helikite aerial vehicle captures and performs texture-based conversion of the aerial images as well as localizing the ground vehicle. The ground robot receives the texture and location information and uses it to plan and execute routes in a closed loop manner using texture difference as a measure of safety.

Results of the tests on the individual components as well as the combined system are presented in terms of speed, optimality and general performance showing the successes and limitations of this approach. A discussion of the results is then presented noting the constraints of the proposed system with suggestions for future work.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Background, goal and motivation

Navigating robotic vehicles in unsafe areas is a complicated problem, due to a lack of access for capturing training data, lack of access for retrieval of robots should problems occur and possible lack of view. These issues are largely due to safety constraints, which present problems for implementing and testing a usable solution.

The goal of this project is to design and implement a system to navigate safe routes using cooperative air-ground robotics. Cooperative air-ground robots have the advantage that they allow both a local and a larger world view by combining data sources and viewpoints.

Originally, the department of computer science here at Aberystwyth university was working in conjunction with geographers to map glacier faces in Greenland. Mapping the face of glaciers is particularly dangerous due to the proximity of falling ice and debris, and teleoperation is challenging due to distance and perspective. The original goal of this work was to research and implement a combined system to assist in mapping glacier faces. The proposed system would navigate along the face of the glacier in real time with the aerial component providing a larger overhead view. The prototype system developed during this work used a wheeled robot in order to simplify development. The work presented here provides a proof of concept, the next step would be to swap the ground vehicle for a boat.

1.2 Research questions and hypotheses

Over the course of the project, the aim has been separated into seven basic research questions:

- **How can ground vehicles be detected by aerial vehicles?**
- **How can a ground-based robotic vehicle be kept in view by an aerial vehicle?**

- **How can positional information be captured with movement in both ground and aerial vehicles?**
- **Which measures for quantizing pixel properties allow effective real-time segmentation of aerial images?**
- **How can paths be categorized as safe or unsafe using pixel values?**
- **Which path planning methods are suitable for real-time use on low powered hardware?**
- **Can a cooperative air and ground system be implemented using the above principles?**

Such questions lead to a series of hypotheses:

- Ground-based vehicles can be detected and kept in view by an aerial platform.
- Real-time segmentation into safe and unsafe routes is possible using aerial platforms.
- Navigation of these routes could be achieved using a cooperative system formed of low powered hardware without training.

The results of each of these research questions and hypotheses are discussed in Chapter 8 with reference to completed work, success, conclusions and opportunity for future research.

1.3 Chapters

This work describes the implementation of a distributed robotic system for capture, classification and traversal of routes without prior training and is organised as follows:

- Chapter 2 covers basic literature on related topics. Path planning methods are discussed with particular emphasis on speed, completeness, optimality and applicability to mobile ground robotics. Cooperative air-ground robotics is covered with particular focus on the types of platforms, cooperation scenarios and practical issues when using cooperative air and ground vehicle teams. Vehicle detection is briefly discussed before discussing texture quantisation focussing on individual methods, typical usage and implementation on lower powered hardware.
- Chapter 3 provides an overview of the proposed system including basic hardware and software composition, distribution of computational tasks and an explanation of how the system functions.

- Chapter 4 explains the design process followed when selecting the aerial and ground hardware. Various options for each type of hardware are proposed with the chosen solution explained in further detail.

- Chapter 5 describes the aerial FlyPi system. Various methods of detecting and keeping above the target are evaluated with an emphasis on real-time usage. A method of detecting the target vehicle is proposed using a pattern based roundel target. Flight parameters and control of the aerial vehicle, a Helikite are also described. A system for real-time stitching and texture conversion of aerial imagery is also explained.

- Chapter 6 describes the wheeled GroundPi unit which receives texture based images from the FlyPi unit. Selection of an appropriate path planner is detailed as well as alterations made to improve performance. Map representation based on an idea of safe texture using an untrained similarity threshold is described as well as the selection of the threshold T value and its effect on path suitability.

- Chapter 7 evaluates the performance of the combined system in terms of speed, accuracy and path optimality in comparison to an ideal path. Tests were initially undertaken indoors using a series of four relative goal locations and different obstacle scenarios. Subsequent outdoor testing took place on multiple surfaces to measure the performance of the system with the addition of the powered Helikite.

- Chapter 8 concludes by comparing the work undertaken to the research goals, evaluates success and suggests further work.

Chapter 2

Background

This section covers a selection of prior work on path planning methods, cooperative air and ground robotics, image analysis and Helikites. Path planning is required in order to safely and efficiently generate the paths needed to navigate the robot. Due to the incomplete and constantly updated view of the world, there is a requirement for real-time updating and re-planning. Cooperative air-ground robotics is the subset of robotics that uses an aerial vehicle to aid ground-based vehicles. Usually, the aerial vehicles provide assistance in either image capture or communication between ground vehicles, which is obviously similar to the cooperation strategy employed in this work. Image analysis briefly describes some methods of robot detection before providing further information on the use of texture as a segmentation domain, as previous experiments using our aerial imagery had been relatively fruitless, the use of colour, in particular, was investigated using various colour-spaces, but in natural images performed poorly. A Helikite was eventually chosen as our platform because of its stability as a tethered platform in the prior works herein.

2.1 Path planning methods

Path or motion planning is a core problem when using mobile robots and has been extensively researched. The selection of a motion planning strategy for a given scenario is a far from easy choice given the large number of constraints including time, vehicle properties, the requirement for completeness, and need for replanning which can vary between approaches. Path planners tend to fall into one of two categories; graph-based or sampling-based depending on their view of the configuration space.

This review seeks to cover a selection of work on motion-planning algorithms with a focus on mobile robot navigation in terms of applicability and relevance to this project. For each planner, their implementation, speed, completeness and typical usage are evaluated. Although

there are a large number of derivatives of each of the main types of planner, the focus is on those that have been used to plan in real life as opposed to having been used only within a simulation. This review is organised as follows, Section 2.1.1 covers various types of motion planners and their derivatives, Section 2.1.2 analyses their attributes whilst Section 2.1.3 provides a general overview with suitability to robot planning and applicability to work carried out in this thesis.

2.1.1 Overview of planners

This work focusses on two main types of planners, graph-based and sampling-based. Other types of planners do exist such as generating Bezier motion curves [1] or splines [2]. These are however rarely used in practice due to their complexity and do not form part of this review.

2.1.1.1 Graph-based planners

The configuration space of the robot is often represented as an occupancy grid or similar structure containing relative safety of the world, including obstacles. By using the information on the state-space, graph search algorithms can be used to find a path from start to goal whilst maintaining an acceptable safety threshold. Many of these graph-search algorithms such as Dijkstra [3], A* [4], D* [5] and D* Lite [6] have been used in previous work on robotic navigation.

Dijkstra's algorithm [3] finds the shortest path between two points, is complete and optimal but is unguided, it grows from source to goal using two lists of points (visited and unvisited), updating the costs and following the lowest points until the goal is reached. Dijkstra is guaranteed to produce a solution with the lowest possible path cost if one exists but is practically slow in large configuration spaces.

A-Star (A) [4]* is an improvement of Dijkstra's algorithm that uses a goal heuristic function that estimates the cost to goal to guide the search. A* is complete and optimal if a suitable heuristic is chosen but cannot partially re-plan.

Dynamic A-Star (D) [5]* is an update of the A* algorithm that allows re-computation of the path given a changing map, likewise being complete and optimal but reducing computation time in dynamic environments by using information from prior plans. D* Lite [6] is a re-implementation of the D* algorithm with improvements in execution speed but with the same result.

2.1.1.2 Derivatives of graph-based planners

Though A* and D* are widely used in their own right, they have both been altered to improve their performance in real-world scenarios.

- Speed and replanning: The original A* algorithm is comparatively slow to plan in large spaces and does not allow for replanning which makes it impractical for online planning in changing environments. Various attempts have been made to introduce replanning to the A* algorithm, the most obvious being D* and LPA* [7] (lifelong planning A*) which use previous search data to speed up future searches. D* itself has subsequently been replaced in most scenarios with D* Lite due to its efficiency with proven equivalent results. D* Lite has in turn been updated using a simplified replanner in order to increase overall speed in the D* Extra Lite Algorithm [8]. In order to satisfy the real-time rule, the ARA* [9] (anytime repairing A*) algorithm was proposed, initially starting by inflating the search heuristic with associated improvements in speed at the expense of optimality and reducing the inflation to converge to an optimal solution where time permits. ARA* was subsequently combined with the D* Lite's ability to replan and the good enough given available time solution of ARA* to form ADA* [10] (anytime dynamic A*). Similarly to ADA*, optimality was also sacrificed in order to permit moving-target planning in real-time with a claimed twelve-fold complexity reduction and minimal (1.5%) reduction in path length.

- Optimality: Graph-based planners despite being touted as optimal typically only use adjacent grid cells when planning paths, at each step, only calculating path cost in the adjacent 8 cells which may not result in the shortest possible path depending on cell resolution. Smoothing can be used on the resultant path [11] but this is not guaranteed to a least-cost solution. A subset of graph-based planners called any-angle planners have been proposed as improvements to both the A* [12–14] and D* [15] searches allowing for any angle paths.

2.1.1.3 Sampling-based planners

Whilst graph based planners are complete and usually optimal, they suffer in terms of time complexity, particularly where there is a large configuration space. Sampling-based motion planners attempt to reduce the time taken to plan by sampling the search space instead of comprehensively covering it, then joining connected samples with a reduction in optimality. Examples of sampling-based planners include PRM [16] and RRT [17–19] and their derivatives.

Rapidly-exploring random trees (RRT) plan a path by growing a tree randomly through free-space but with a bias towards the goal by selecting points and then connecting them if possible. RRT has been proven to be probabilistically complete but in its raw form is non-optimal.

Probabilistic roadmaps (PRM) are a class of sampling-based planner that select a number of random configurations and then attempt to connect them. PRM consists of two phases, preprocessing and connection. The PRM must be used in conjunction with another motion planner, dependent on the application in order to connect the nodes.

2.1.1.4 Derivatives of sampling-based planners

Because the PRM is more of a design than a rigid planning algorithm, many of its derivatives are simply combinations of preprocessor and connector [20].

The basic RRT algorithm has various disadvantages, namely that it is non-dynamic, has no concept of path cost and may become stuck in local minima when present. Subsequent alterations have been made to the core algorithm to counter RRT's shortcomings:

- Speed: Multiple speed improvements were suggested, the simplest being introducing bi-directional trees which may help to ensure quicker convergence and escape local minima [21]. The eRRT algorithm attempted to guide search using prior information from previous searches [22]. Other improvements mainly biased the search toward the goal in order to reduce search time [19, 23].
- Optimality: The original RRT is not optimal and simply tries to connect the start and goal point via a random path. RRT_OBST (rrt obstacleness) places a limit on traversability, increasing this if a path is not found [24]. Transition RRT [25] integrates a self-tuning cost-transition test function in order to prioritise lower cost areas. The RRG [26] (rapidly exploring random graphs) and RRT* [26, 27] find a low-cost path by considering cost when extending the tree. RRG uses a graph-based structure, whereas RRT* improves on this by returning to the tree structure.
- Dynamicity: Dynamic-RRT [28] attempts to allow replanning by pruning and regrowing nodes.

2.1.2 Planner attributes

2.1.2.1 Completeness

As a basic requirement in most scenarios, planners are required to be complete i.e. finding a path if one exists. Two types of completeness are prevalent, resolution where the planner will

be guaranteed to converge to a solution and probabilistic where the likelihood of not finding converges toward zero given an increasing number of runs.

All of the graph-based planners tend to be resolution complete which varies little between earlier and later works. Sampling-based algorithms by their very nature are not guaranteed to be resolution complete as they don't intensively sample all or part of the configuration space but tend to be probabilistically complete. The difference between resolution and probabilistic completeness in practice is likely to be minimal where there is an acceptable sampling strategy, time and number of potential paths. Achieving probabilistic completeness would likely negate the speed advantages of the sampling-based planner, requiring large areas of complete coverage.

Instead of focussing on completeness, most work has focussed on improving the execution speed of the planner. In the case of graph-based resolution complete planners, it will either be found or not found in less time based on a heuristic or informed search such as D*. In sampling-based planners, biasing and heuristics have been used to improve the convergence speed/likelihood given N cycles.

Completeness requirements for mobile robots are largely dependent on the scenario. Given enough time and a sensible enough scenario, sampling-based planners are likely to converge to a solution, especially with improvements to bias growth towards a goal and escape local minima.

2.1.2.2 Optimality

Optimal path planners are planning methods that will always find the lowest cost path between any two points. Generally speaking, whilst it is possible to use abstract concepts such as safety as the path cost, distance travelled is perhaps the simplest and most frequently used path cost metric.

Graph-based planners such as Dijkstra, A* or D* claim optimality, however, due to the need to sample all graph cells are costly in terms of speed. Whilst graph-based planners will provide an optimal solution, the use of a cell-based graph means that this optimality only extends to finding the most optimal path available using the cost of adjacent cells which will be limited to increments of 45 degrees (0,45,90,135,180,215,270,315) between these adjacent cells. To counteract the limitations of planning using adjacent cells, a selection of any-angle planners were proposed [12–15]. Post-planning smoothing [11] of graph-based planners was attempted with a comparatively low increase in execution time but no guarantee of absolute optimality. Theta* [12] outperforms path smoothing in terms of achieving a lower path cost but is typically 5-10 times slower than the traditional A*. Whilst Block A* [13] typically performs similarly to Theta* in terms of time taken, it does require extra time to generate a

local distance database, which may slow down overall performance. Interestingly ARA* [9], ADA* [10] and [29] all bucked the trend by reduced their performance in order to gain speed with [9], and dynamically increasing performance as time permitted to produce the lowest possible cost path available for the time allowed.

Sampling-based planners are almost exclusively non-optimal in their base form due to the lack of a comprehensive node evaluation strategy. Similar to any-angle planning, the simplest method of path optimisation is to implement post-planning smoothing, both in terms of speed and simplicity of implementation with an associated reduction in cost but no guarantee of optimality. As an alternative to path smoothing, the RRT algorithm has been extended to incorporate path costs in RRT_OBST [24]. Whilst this provides an easily-implementable solution, its weakness is that there is no method to decrease the cost threshold once increased and therefore it is unlikely to converge to an optimal solution unless given sufficiently small threshold steps and a uniform increase in cost towards the target. Multiple RRT_OBST trees were also proposed improving the optimality but this is dependent on number and distribution of trees being sufficient in the configuration space [24]. Transition RRT attempts to increase optimality by prioritising growth towards low-cost regions using a transition test and self-adapting temperature, is not optimal but has a decreased cost in comparison to the original RRT [25]. The RRG [26] and RRT* [27] differ from other work in that they claim to be asymptotically optimal, that is that they will almost certainly find an optimal solution but at the cost of a large number of iterations which may limit optimality in real-world conditions.

The requirement for optimality should be judged carefully when implementing a robot planner, in many real-world scenarios where there are multiple possible paths a “close enough” solution may be preferable in terms of the advantages gained in processing speed. Graph-based planners such as A* or D* clearly have an advantage over sampling based in terms of almost universal optimality when using adjacent cells. However, in their original implementation most-graph based planners do not achieve any-angle planning, in contrast to the sampling based strategies. Both the sampling and any-angle graph-based planners did see path cost improvements when using post-planning smoothing but without full optimality. Improvements suggested to each type of planner such as Block A* [13] or Transition-RRT [25] to improve path costs did seem to reduce the path cost but at the cost of computational complexity and still without a guarantee of true optimality.

In real-world scenarios with dynamic environments and a need for replanning in real-time, optimality is often sacrificed in favour of a fast but good enough solution such as an RRT with path cost improvements or ADA* with a limitation on the processing time, and therefore optimality.

2.1.2.3 Speed

The speed of each planner is largely dependent on the sampling strategy, its computation and the size of the configuration space as well as the hardware.

Graph-based planners such as Dijkstra or A* typically evaluate a large area of the configuration space and as such the time is likely to violate any real-time requirement on low powered computational hardware. Dijkstra is uninformed and as such is likely to converge slowly towards a solution, whereas A* and D* are informed which tends to result in quicker convergence to a solution. Alterations to improve speed on graph-based planners have tended to either be implementation improvements such as D* Lite [6] or to save time during updates by using prior information as in the LPA* [7]. Most interesting are the ARA* [9] and ADA* [10] algorithms, these have the advantage for real-time use that they start off by inflating the heuristic to find a quick sub-optimal path and then iteratively improve the result by deflating the heuristic back towards optimal as time permits.

Sampling-based methods are typically fast in comparison to graph-based methods due to their evaluation of minimal areas of the configuration space. Problems arise with sampling-based methods where they become trapped in local minima, which can either slow or prevent the planner finding a path or when they fail to bias tree growth sufficiently toward the goal. In order to prevent local minima trapping, a bi-directional search strategy was proposed [21] which converged more quickly toward a solution. Alternative approaches to improving speed included assisting tree-growth using previous searches [22] which is analogous to partial re-planning, and biasing towards the goal [19].

Speed is often crucial when selecting a planner. Graph-based methods are typically best suited where a completely optimal path is required because of their guaranteed optimality. Where real-time planning is required and non-optimality is acceptable, the sensible choice appears to be a sampling-based method. Of the sampling-based methods, the RRT and its derivatives are likely to be the fastest to achieve a good enough path.

2.1.2.4 Replanning

Of the sampling-based planners, most do not allow for replanning in their original implementation. Replanning can be either local or full depending on whether a subset or all of the map has changed. Lack of replanning is particularly an issue where there is incomplete data or where the environment changes unpredictably during motion.

D* and D* lite are the only graph-based planners to implement local replanning as standard using the assumption that free space is empty and replanning as updates are received. LPA* [7] doesn't strictly locally replan but fully replans whilst implementing reusable parts of the

tree. The likelihood is with graph-based planners in real-world robotics situations that the requirement for a replan will exist. If a replan is needed it must be achieved quickly enough that the updated map is not redundant by the time the path is replanned.

The original RRT planner doesn't facilitate local or partial replanning. The Dynamic-RRT [28] derivative implements replanning by pruning blocked paths and re-growing the original tree to connect the current location and goal. In practice, the pruning and re-growing strategy simply consists of a collision-checker and the ability to prune and reattach a new path so can be easily implemented in addition to any of the other alterations to the basic RRT algorithm. Unlike the graph-based planner where there is likely to be a large cost of fully replanning, the overall speed of the RRT and its derivative algorithms may negate the requirement for local replanning given the comparatively low cost of a full replan.

Both graph and sampling-based planners don't implement local replanning by default. Replanning in the context of mobile planning will either take place where the robot has moved into an uncharted area and there is an obstacle, or the mapped area has changed. The graph-based planners all suffer from the same problem in that where there is a large configuration space or distance between goals they are likely to fail to update and fully replan quickly enough, though local replanning is possible in LPA* and D*, whether or not they are suitable is highly dependent on the size of the configuration space and computational power. Sampling-based planners, on the other hand, have the advantage that they can be quickly pruned and regrown and so are more likely suitable for rapidly-changing environments where the complete map is not known before execution.

2.1.3 Discussion

Of the generally accepted robot planning methods described above, it is firstly immediately clear that some are little used in the current literature. A*, for example, has generally been replaced in practical usage by D*, which in turn has been replaced by D* Lite due to its speed and ability to save time when replanning using prior information. Of the sampling-based methods, the general trend is to use an RRT derivative because of its flexibility and simplicity over methods such as the PRM. In terms of completeness, A* and its derivatives such as D* are the most commonly used where path optimality is crucial, such as a lunar rover but where the environment does not contain a large amount of changing obstacles. Where there is a real-time requirement and a good enough solution is acceptable, sampling-based planners such as PRMs or RRTs are more suitable. Though sampling-based planners are not guaranteed to be complete, given a suitable time (which would more than likely be less than a complete planners time to find a plan), they are likely to find a solution.

Graph-based planners such as A* and D* based on Dijkstra's algorithm are pretty much

universally complete as they will sample large parts of the configuration space. In most cases, however, a good enough solution would be acceptable for navigation rather than a completely optimal solution, with a preference being given to other factors such as planning time. Of note are the ARA* and ADA* algorithms that are graph-based but produce a most optimal path for the time given which may be more suitable for real-time applications. The sampling-based planners are by definition non-optimal, the most popular RRT has been altered to incorporate path-smoothing, bi-directionality and cost of traversal which are likely to produce a more acceptable closer to optimal solution that satisfies the real-time requirement. In practical terms, the RRT and sampling-based planners have the advantage that their connection strategy allows for any-angle planning, which may actually be more optimal in some cases than a standard graph-based planner such as A* which cannot.

The graph-based planners are by definition slower than sampling-based methods as they sample much of the configuration space but produce optimal results. By improving Dijkstra's algorithm, they are now guided by heuristics which allows them to converge to a solution faster, may purposely trade optimality for speed, and may incorporate the use of previous methods to replan which is necessary for real-time applications. Sampling-based methods, however, may not replan by default but converge to a solution significantly faster and so for practical usage, this may not matter. A common strategy for replanning is the incorporation of a node pruning and reconnection strategy, this has been demonstrated in multiple works where incomplete data necessitated replanning during execution. For real-time applications, sampling-based planners clearly have the advantage in terms of speed but at the cost of guaranteed optimality but with a lower cost overhead in terms of storage space. Graph-based planners, however, guarantee optimality but at the cost of execution speed though in general aren't suitable for any-angle planning.

2.2 Cooperative air-ground robotics

The ready availability of low-cost aerial and electronic hardware and miniaturization of electronics in recent years has meant the research community has increasingly focussed on the use and development of aerial platforms. Flying hardware can provide a method of providing data that would otherwise be unavailable for reasons such as safety. Whilst aerial robots allow a larger field of view than ground-based robots, they are disadvantaged in that they may not be able to perceive smaller objects or obstacles. Conversely, ground robots have a good view of smaller and closer objects but likely lack a bigger picture view. Combining ground and aerial vehicles together (cooperative air-ground robotics) is a comparatively new field but gives the advantage of both a short-range and long-range view. This section describes the current state

of the literature on cooperative air-ground robotics but is limited to objective, the platforms types and formations, the type and level of communication between the platforms, localisation and the issues encountered. Reference is also made to the environment in which the platforms operate. The context of the review is suitability for navigation in unknown areas and similarity to work completed in this thesis.

2.2.1 Cooperation scenarios and objectives

Within the existing literature, there are three primary cooperation scenarios for air-ground robotics depending on the parameters of the scenario and the type of data fusion from aerial and ground components:

- *Aerial robots assist ground robots (AAG): The flying vehicle assists the ground vehicle by providing imagery, data and or locational updates.*
- *Ground robots assist aerial robots (GAA): The ground vehicle provides locational or movement data to the aerial vehicle in order for it to safely follow or land on the target vehicle.*
- *Ground and aerial robots cooperate (GAC): A combination of the flying vehicle providing imagery, data or locational updates and the ground vehicle providing locational or movement data.*

Although other scenarios do exist, using combinations of multiple levels of seniority of ground robots [30], these still fit broadly into the above categories.

Cooperative air-ground robotic teams have been shown and theorised to be useful in a variety of real-world situations, with multiple objectives. Path Execution is the prevalent objective [31–38], the aim being to transport the vehicle and potentially cargo [39] in between two locations. Other commonly used objectives include using the cooperative team to complete tasks in dangerous areas [30, 40–44] or to provide surveillance [45–47]. Swarm control [48] and assisted landing [49, 50] are other scenarios that have involved the explicit cooperation between both air and ground robots.

2.2.2 Components

2.2.2.1 Platform types

Using the cooperative model, there are various different types of platform including unmanned aerial vehicles (UAVs), unmanned ground vehicles (UGVs) and unmanned underwater vehicles (UUVs).

Rotor-based UAVs such as the quadcopter or helicopter have the advantage that they are typically small, have a wide altitude range and are able to hover in a single location or move quickly as required. They are however limited in their longevity and payload. Blimp based UAVs have the advantage of longevity and stability but are not as controllable as other types of UAV and must be tethered [51]. Little work appears to have used aeroplane based UAVs possibly due to their lack of ability to maintain position and keep a target in view. Work using aeroplane-based UAVs that has been completed consists of cooperative surveillance, which may not need the ground vehicle constantly in view [45]. UGVs have the advantage that they are able to interact with ground objects and have a good quality view of the local area but lack a global view which may lead them to become trapped or taking sub-optimal paths. UUVs are more difficult to use because of the unpredictability of water currents and difficulties in using wireless methods underwater, such as GPS, radio or wifi [52]. Kites have been used in various works, though not often in conjunction with a ground vehicle. The IKAPP [53] used a similar low-powered computer and mounted camera. The kite itself was very stable and suitable for high-altitude flight but requires relatively stable wind and a comparatively high flying height, whilst also sometimes suffering from motion-blur. [53] did note that whilst there were other possible sources such as satellite and imagery from manned flights, the update frequency made these impractical for real-time usage.

2.2.2.2 System architecture

Regardless of actual robots used, the layout of the system varies hugely between works. A single UAV and a single UGV are a common combination, which limits the complexity but may also cause problems should the vehicle go out of view for any period of time. Alternatively, [32, 54] supplemented the single UAV and UGV with ground-based base stations, these act as an intermediary between the aerial and ground vehicle, allowing them to perform more computationally expensive tasks. Obviously, in this configuration, the range of the robot is constrained by the base stations. Swarm control of multiple robots by a single UAV is exhibited, these, however, tend to only provide ideas of the location of a single UGV [43, 55] or an idea of swarm state [48] as opposed to exact locations or controls for individual vehicles due to the complexity, with individual control being due to swarm behavior. Multiple UAV's provide a larger field of view than a single robot with increased complexity in terms of data fusion and control, especially in real-time. Various multiple UAV architectures were proposed, the simplest being multiple UAV's communicating with single robots, extending to multiple robots. A more complex strategy of using a command UAV (blimp) with multiple sub UAVs (quadcopters) was demonstrated that provided information to ground vehicles [30]. Whilst multiple vehicles were used in various scenarios, these tended to be limited to non-navigation

tasks such as fire detection [30] and in particular cooperative surveillance of an area [46] due to the increased field of view. These systems did not need to exactly follow an individual ground vehicle. Singular UAV solutions tend to be required to follow the target vehicle for navigation or other purposes.

2.2.2.3 Capture, transmission and data processing

Capture of data from a UAV usually consists of colour images, except in specialist applications such as fire detection where an infrared camera can also be used [30]. Most UAVs found in the literature had a single camera, however, there were examples that comprised multiple cameras which allowed the vehicle to maintain the UGV following whilst capturing the most appropriate field of view [50]. Data transmission is computationally expensive as is complicated image processing on low-powered hardware. In many cases the aerial platform is simply used as a robot-following camera, transmitting imagery to the ground or intermediary computer [32, 54] for further processing, particularly where there is only a single UAV/UGV pair. Where multiple ground-vehicles were involved, the data processing tended to take place before transmission in order to ascertain the basic shape of the robot group or swarm [48, 56] In practice, direct UGV to UGV communication was rarely used, with most using base-stations as an intermediary in order to facilitate communication between multiple vehicles.

2.2.2.4 Localisation of vehicles

Each of the robotic components, UGV or UAV should have some idea of its current location, be that in real-world coordinates or relative to another vehicle. In order to ascertain real-world location, at least one of the component vehicles will need to know its real-world location or location in the context of the scenario. When detecting a vehicle to be followed, there is an implicit requirement to detect its location as quickly as possible so that the maximum number of control steps can be taken. Various approaches exist, typically consisting of image rectification, detection of the ground vehicle in the frame and fusion of aerial data with ground data. Real-world data capture, particularly using imagery from an aerial vehicle introduces its own challenges, the possible lack of matching features between frames, including changes in scale (height), rotation and changes in lighting due to weather and shadows.

Measuring the absolute location of both ground and aerial vehicles typically uses a multitude of data sources in addition to captured imagery including:

- GPS (Global Positioning System): Provides global location using a series of satellites with of an accuracy of approximately five metres. Alternatives include GLONASS (Russia), Galileo (Europe) or Beidou (China).

- RTK-GPS (Real-time kinematic GPS): Use of multiple GPS receivers, one with a fixed known location to improve the accuracy of a GPS signal.
- IMU: An electronic device that can be used to measure altitude and velocity.
- Odometry: Use of the robots own sensors to estimate the change in position over time

In many situations, GPS or similar technologies are unsuitable because of poor performance within and near buildings, interference and lack of resolution. IMU and Odometry information can also be unreliable. Various fusions of data from both the aerial and ground platforms exist, a large number of them using the Extended Kalman Filter [57, 58] in order to fuse locational inputs from multiple sources.

2.2.2.5 Issues

Longevity is a particular concern in air-ground robotics, particularly where there are multiple interdependent vehicles, relying on each other for data. Robotic vehicles with motors such as helicopters or quadcopters have very short flight spans requiring frequent charging [37]. Blimps, on the other hand, have longer flight times due to the lack of a need to use power but require tethering to keep them in position. To avoid costly recharging, various works have used a power tether [37, 42] to increase longevity and have implemented a self-retracting and extending smart tether system [59].

Latency between, detection, transmission, receipt and action was detailed in multiple works, typically this was due to the high cost of transmission, whether wired or wireless, particularly where there are intermediary base-stations or computers. The effect of latency was generally to delay localisation of the robot itself causing over-driving or turning. Where latency was mentioned, the delay was typically combatted by using an estimate of the time delay and a prediction of world model within that time to preempt its effect on the movement of the ground vehicle.

2.2.3 Discussion

Much work has been completed on the use of teams of aerial and ground robots to overcome difficult situations. Most work has taken advantage of the FOV advantages of the aerial robot, though in some cases there is some feedback to assist with the following of the UGV. The most common architecture seems to be a single UAV and UGV pair, though other architectures were proposed, they tended to be constrained by the complexity of calculation and transmission of data to multiple agents simultaneously.

The majority of prior works use conventional cameras, except where there is a specific use for more specialised equipment such as to aid fire detection. In order to localise the vehicles, multiple data sources such as GPS, IMUs and odometry are often combined to provide more accurate data than using a single sensor.

In particular, previous literature tends to lack real-world testing and information on problems such as occlusion, loss of connection, loss of tracking and longevity, which have been discussed little. Longevity is a particular problem, particularly using power intensive flying robots.

2.3 Image analysis

This section details previous works on the use of texture as well as localisation of ground vehicles in the context of cooperative air-ground robotics. At the start of this project, initial work investigated the use of colour not only as a method of target localisation but also as a method of classifying areas of captured imagery. Colour was used, starting with binary, then more complicated thresholding in order to attempt to classify safe and unsafe areas. In addition to the RGB colour space, other spaces such as HSV were also investigated, particularly to try and avoid the effects of shadows and changes in light levels. Unfortunately, because of the complexity of natural images, colour proved to be ineffective and lacked detail, particularly where there were changes in light levels, shadows or even where the same surface was subtly different colours e.g. a road surface of varying ages of tarmac.

Texture can be thought of as the spatial arrangement of colours within an image or region of an image and has been used extensively in computer vision as a method for classifying or segmenting regions of images based on their inherent properties as an alternative to or in conjunction with other models such as colour. Texture was chosen because of the increased level of information available, in particular, its ability to differentiate between surfaces in natural images.

Texture was selected as a measure with the premise that as texture is usually relative to the coarseness or changes in colour intensity over a surface. If a “safe” area is known then textures that are similar will have a similar coarseness, and also likely be safe to traverse. Texture also has the advantages that since it uses the relation of pixel intensities rather than the actual values, it is likely to be more invariant to changing light levels than a purely colour based system.

In addition to quantification of aerial images, existing methods of vehicle localisation were investigated, particularly those that had been used in the context of cooperative air-ground robotics in natural environments.

Given the requirements of the project to implement a solution that navigates unknown areas, there is a need for real-time localisation and quantification. Because of the requirements of the project, investigation of both texture and localisation in previous works focus on the use of lower powered hardware, in real-time and in natural environments.

2.3.1 Types of texture measure

Texture can be thought of as the spatial arrangement of colours within an image or region of an image. To gain textural information from a region, there are various main approaches:

Structural methods attempt to define the spatial arrangement through a series of equations describing a repeated pattern using property and placement rules. In practice, these are rarely used in real-world situations due to complexity, and inflexibility with unpredictable scenery or motion. Examples of statistical methods include methods that decompose structure into a series of shapes, patterns and placement methods [60] or by describing real-world textures as a transformed a series of ideal textures [61].

Statistical methods measure the number of particular arrangements of features within an image or region. Examples include GLCMs [62], LBPs [63] and Law's texture energy filters[64]. In practice, statistical measures are used in classification and segmentation of everything from cancer to aerial images and are usually comparatively simple to calculate in comparison to other methods given their simple mathematical nature.

Model-based methods attempt to construct an image model in order to quantify and synthesise texture. Examples of model-based methods include Fractal models [65, 66] and Markov Random Fields [67, 68]. In practice, the cost of calculation of these models makes real-time texture segmentation difficult. Model-based methods have been used on aerial imagery [69] with success in unsupervised segmentation, though little reference is made to execution time with most work being offline.

Transform-based methods use transforms such as the Fourier [70], Gabor [71] and Wavelet [72] to transform the textural images into a set of textural characteristics that can be compared by applying banks of filters to the image in order to generate these characteristics. Both wavelet and Fourier transformations have been used in prior work on aerial images [73, 74] but these are both offline and don't mention execution speed. real-time texture transformation using Gabor filters was also proposed [75] but this was not applied to large-scale aerial images. As with model-based methods we expect transform-based methods to be too slow for practical use in real-time low power robots.

Typically structural methods are more useful for regular or repeated patterns e.g. wallpaper where a pattern exists, statistical measures being used in unconstrained real-world scenarios. Statistical methods have been used in a large breadth of work, with attempts at speed im-

provement and online generation and flexibility applied in many different fields. The most commonly used of the statistical methods is the GLCM and textural features derived from it in order to quantize texture.

2.3.2 Statistical texture measures

Since statistical texture methods have been used successfully for a range of different applications, and enable at fast and real-time calculation. Therefore these methods have shown the most promise for the current project. Different statistical methods are described below.

2.3.2.1 GLCM

The most cited method for extracting texture is the grey level co-occurence matrix (GLCM). Grey level co-occurence matrices are based on the quantization of pixel intensity values at pre-specified relative locations [62], an example of the matrix generation is shown in Figure 5.4. GLCMs in their raw form do not provide textural information but can be used in conjunction with a series of fourteen textural calculations (Haralick's texture measures) [62, 76] that quantize textural properties. The selection of appropriate singular or multiple Haralick texture quantities depends entirely on the type of image scene, with large numbers of possible combinations existing for the 14 measures. Contrast, Correlation and Entropy were suggested as a valid combination of measures [77] and have been extensively used. Entropy, difference variance, difference entropy, and difference average were proposed as working well individually whereas energy, contrast, correlation, inverse difference moment and mean worked in combination [78].

The original GLCM is not invariant to scale or rotation but has subsequently been extended to allow for changes in rotation [79] and scale [80]. The original GLCM is computationally slow due to the need to parse the whole image in order to populate the frequency matrix which will more than likely be sparse and may require calculation in multiple orientations/scales. In terms of simplicity, the number of channels can be reduced to decrease the size of the GLCM. Stepping across image pixels with the assumption that local pixels are likely to be similar speeds up the method but requires the selection of the additional parameters in addition to the relative distance and orientation of the base GLCM [81]. A linked-list structure (GLCLL) was proposed by Clausi and Jernigan, this replaced sparsely populated matrices with a sorted linked list [82]. Svolos and Todd-Pokropek improved upon the GLCLL by replacing the linked-list with a tree structure [83], reducing the time taken to search significantly. The GLCHS (grey level co-occurence hybrid structure) [84] combines the linked list of the GLCLL with a hash table, thus avoiding the need for sorting the linked list and is used as part of the

GLCIA (grey level co-ocurrence integrated algorithm). GLCIA improves the speed of the GLCM by combining the GLCHS with a new method of combining the GLCHS framework with sum and difference histograms (GLCHH) [85]. Unfortunately, GLCHH cannot be used to calculate some of the measures of texture and so is used in combination with the GLCLL where needed. The time taken to calculate features using the GLCIA varied between 0.04 to 16% of the same task on the GLCM depending on quantization levels and selected features. Improvements in speed have also been suggested by the use of specialised processing hardware, with a 4.75 fold increase in speed versus the original algorithm [86] on non-specialised hardware. The original GLCM has been trialled on lower powered hardware with slow results in comparison to conventional desktop computers [87].

2.3.2.2 Local binary patterns

Local Binary Patterns (LBPs), are a method of texture quantification that functions by thresholding the neighbourhood pixels (typically 3×3) of a central reference pixel, using the value of the central reference pixel as the threshold value and considering a string formed of the thresholded neighbour pixels as a descriptor [63], an example of which is shown in Figure 2.1. The original LBP descriptor is not rotation or scale invariant, the uniform patterns extension [88] implements rotation invariance by introducing the idea of common "uniform" patterns where there at most two transitions in the binary pattern (0 to 1 or 1 to 0). Whilst computing the histogram, those uniform patterns are grouped separately in bins from non-uniform patterns (such as the 01010100 from Figure 2.1), which are all grouped in a single bin, reducing the number of bins in the histogram. Also proposed were multi-scale LBPs using any distance from the centre pixel and bilinear interpolating values. LBPs are most typically used in close range applications such as face or iris recognition, though they have been used previously with aerial images [89].

2.3.2.3 Law's texture energy filters

Law's texture energy filters are a series of local masks proposed in 1980, sixteen 5×5 pixel masks exist as the outer product of pairs of four vectors representing Level, Edge, Spot and Ripple [64], with certain symmetric pairs being removed, this equates to nine measures of texture. After preprocessing the initial image, the masks are applied to give a series of nine texture values for each pixel. In practical usage, little work has recently been completed using these measures, them having been surpassed by other statistical methods.

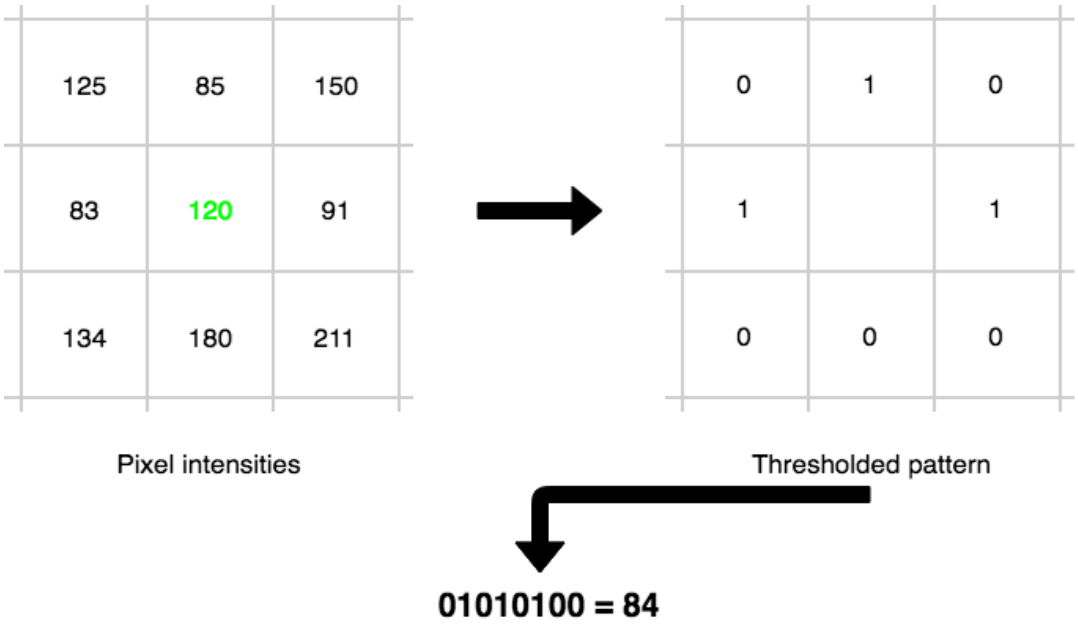


Figure 2.1: LBP generation example

2.3.3 Localisation of co-operative vehicles

When discussing detection, It is important to note the distinction between detection and tracking (which are interchangeable in normal usage), detection being the independent finding of an object in a single frame and tracking being the identification and following of a moving target through multiple consecutive frames. Much existing and recent target detection uses stationary cameras over multiple frames (examples include frame differencing, background subtraction and template matching), these tend to be computationally simple but are unsuited to cooperative robotics due to the requirement for a static viewpoint. The Camshift derivative [90] of the Mean-Shift [91] algorithm has the potential to track using a non-static viewpoint but is computationally unsuitable as part of a Raspberry-Pi based system due to maintaining a confidence map in real-time.

In the main, feature-based methods were discounted because of the poor performance on the low power hardware, though were been subsequently used later. Of the feature-detection methods, some such as FAST [92, 93] are not particularly resilient to noise, which will be problematic in a varying outdoor environment, some such as SIFT [94, 95]or KLT [96, 97] are too slow and some such as SURF [98] are proprietary. ORB [99] is a fusion of the FAST detector and BRIEF [100] descriptor with performance improvements, and has been shown to faster than SIFT/SURF and so is more useful on low-power hardware, like the Raspberry Pi hardware which was subsequently chosen for this project.

Computationally simpler works typically use pattern or colour based matching such as a

system of multicoloured LEDs [48, 49, 101] allowing simple pose, scale and heading. Other methods use a series of circles and rectangles [58], a fiducial tag [102] or a coloured pattern target [103]. McIntyre, Church and Labrosse [104] proposed a tracking system using an image of the desired target that was transformed in subsequent images in order to localise the target using a framework, this was applied to a similar "eye in the sky" scenario, however this wasn't demonstrated in natural environments with a rapidly changing viewpoint. Recent work on this has tended to have been undertaken by hobbyists due to the advances in more computationally complex variable target solutions[1][2] . Depending on height above target, some of these targets may not be visible or recognisable, interestingly at least one work uses optical flow as a substitute for simpler target detection where target detection fails due to a low height [49].

Loss of the ground vehicle by the aerial vehicle is not explicitly defined in many of the prior works, whether or not this is because of the excellent performance of the ground vehicle tracker or lack of solution is not clear. Many of the works documenting cooperative air-ground robotics either use quadcopters, helicopters or blimps. Quadcopters and helicopters have the advantage that they are fast and have large numbers of degrees of freedom, potentially given a sufficient field of view and speed may mean that the likelihood of target loss is minimal. Some works did implement a backup estimation of location using optical flow when the flight height is too low for conventional target detection.

2.3.4 Discussion

Of the different types of methods to generate texture, there seemed to be four main types of method that exist in prior work. For the purpose of this project, there are three overriding concerns which are the performance on overhead imagery, applicability to real-time generation and generation on low powered hardware. For real-time, real-world usage, structural is not prevalent or particularly useful due to its complexity and preference for generated or repeating textures which are unlikely in real-world environments. Statistical measures have been used in a large body of work, most commonly the GLCM and its derivatives in conjunction with Haralicks texture features in different combinations to generate texture. Though the original GLCM is expensive in terms of generation time, subsequent updates have been proposed to improve speed and decrease execution time. Improvements in execution speed may enable this to be run in real-time on lower powered hardware, though there appeared to be little work on real-time usage on low powered platforms. Local binary patterns are typically used for

[1] Technical-Recipes. Tracking Coloured Objects in Video using OpenCV. URL http://www.technical-recipes.com/2011/tracking-coloured-objects-in-video-using-opencv/ (Accessed 28/12/2016).

[2] Adrian Rosebrock. Detecting Barcodes in Images with Python and OpenCV URL http://www.pyimagesearch.com/2014/11/24/detecting-barcodes-images-python-opencv/ (Accessed on 31/12/2015).

fine-scale textures and have largely not been used for aerial images.

The localisation of ground vehicles in real-time from a moving vehicle typically used computationally simple methods such as brightly coloured circles, patterns or fiducial tags in order to satisfy the real-time requirement. Unfortunately, much of the literature lacks detail particularly regarding target loss, though interestingly there were attempts to use other methods (optical flow) where there was a known likelihood of non-detection at low flying heights.

2.4 Helikites

Prior work on Helikites has tended to be fairly minimal given their relative obscurity. Verhoeven et al [105–108] used the platform to undertake aerial surveys for archaeological purposes. Dougherty et al [109] similarly attempted to measure the condition of grasslands. More recently Fonstad et al [110] attempted structure from motion and Young-Heon et al [111] attempted bathymetry (measurement of water depth). Aside from using the platform to capture imagery, the platform has also been tested for deploying wireless communication equipment [112, 113] and further work on this is ongoing under the auspices of the EU-funded ABSOLUTE project[3]. A similar Elevated Balloon-Kite Hybrid (EBKH) was also developed [114] though in practice, this appears to vary little from the Helikite platform.

In the main, the literature on Helikites has tended to use larger and unpowered platforms, Verhoeven [105–108] used a $7m^3$ variant whereas Dougherty [109] and Young-Heon [111] used a smaller $3m^3$ variant. Fonstad [110] used a $1.6m^3$ variant similar to ours. Of the previous work on Helikites, Verhoeven [106], Dougherty [109], Fonstad [110] and Young-Heon [111] used consumer digital cameras, Verhoeven mounted his using a picavet suspension. All of these processed the imagery offline, after flight.

2.5 Conclusions

Having looked at prior work on route planning, cooperative air-ground robotics, relevant image analysis and Helikites, it seemed appropriate to select techniques from those presented herein for further investigation in this project. RRTs particularly lend themselves to this project because they are fast, low-cost and adaptable to allow replanning which is necessary given the incomplete nature of incoming data. Of the cooperative air-ground robotics literature, inspiration was taken from tethered systems and blimp-based platforms to satisfy longevity and stability requirements. To determine texture, GLCMs and Haralicks texture

[3] ABSOLUTE consortium. FP7 Absolute project. url: www.absolute-project.eu, 2015 (accessed 05/20/2016).

features seemed like the obvious choice due to their flexibility and simplicity of calculation. Helikites are relatively little-used in prior literature but have been shown to be stable as both camera and communications platforms, typically they are flown at a much higher altitude than this project but have not so far been powered in other works.

Chapter 3

System overview

3.1 Scenario

The initial project was to implement a distributed system of navigating robotic vehicles in unsafe environments. The project eventually evolved into implementing a co-operative air and ground system based on texture difference as a measure of safety for navigation.

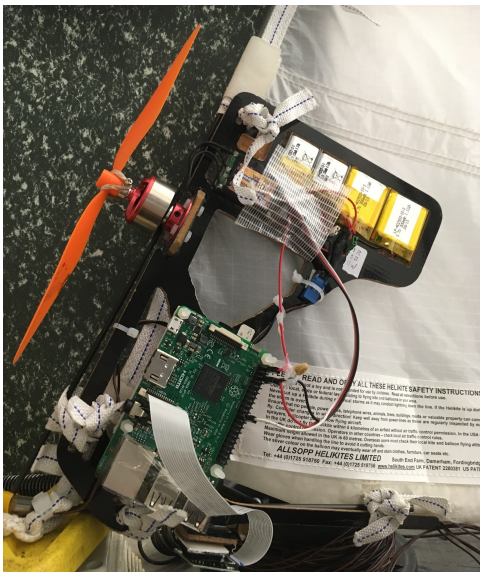
3.2 Requirements

The original brief envisaged a flying camera platform with some form of transmission to a ground-vehicle controller enabling navigation and movement. The crucial design decision was the selection of a Raspberry Pi computer and its camera as opposed to a "dumb" camera and transmitter allowing for more balanced task separation and fulfilling the following requirements:

1. Load balancing: The processing load should be evenly balanced such that one component is not normally waiting for the other for extended periods of time.

2. Minimal communication: Since communication is expensive in terms of time and power, communication should be reduced to a minimum.

3. Redundancy: Because of the unsafe nature of the environment, the components should have some redundancy in terms of ability to work separately and regain communication in the event of a communication loss or error.



(a) FlyPi unit

(b) Allsopp Helikite

Figure 3.1: Flying vehicle hardware

3.3 System structure

The system itself consists of two components, FlyPi and GroundPi.

FlyPi (See Figure 3.1a) is a Raspberry Pi computer and camera with a connected speed controller and brushless motor. The FlyPi unit is mounted to an Allsopp Helikite (See Figure 3.1b) and provides a top-down view of the area under the Helikite.

The Helikite and FlyPi are attached via a combined power and security tether to the ground vehicle; a Pioneer P3-AT was used in the tests. The GroundPi unit is another Raspberry Pi based computer that interfaces wirelessly with the FlyPi and sends movement commands to the P3-AT over ethernet to control vehicle movement.

An overview of the system is shown in Figure 3.2.

3.4 Hardware components

The FlyPi components are detailed in Table 3.1, they are primarily off the shelf components apart from the carbon fibre mounting board which was custom made to fit the Helikite.

The GroundPi simply consists of a Raspberry Pi V3, attached to a Pioneer P3-AT mobile research robot (Figure 3.3).

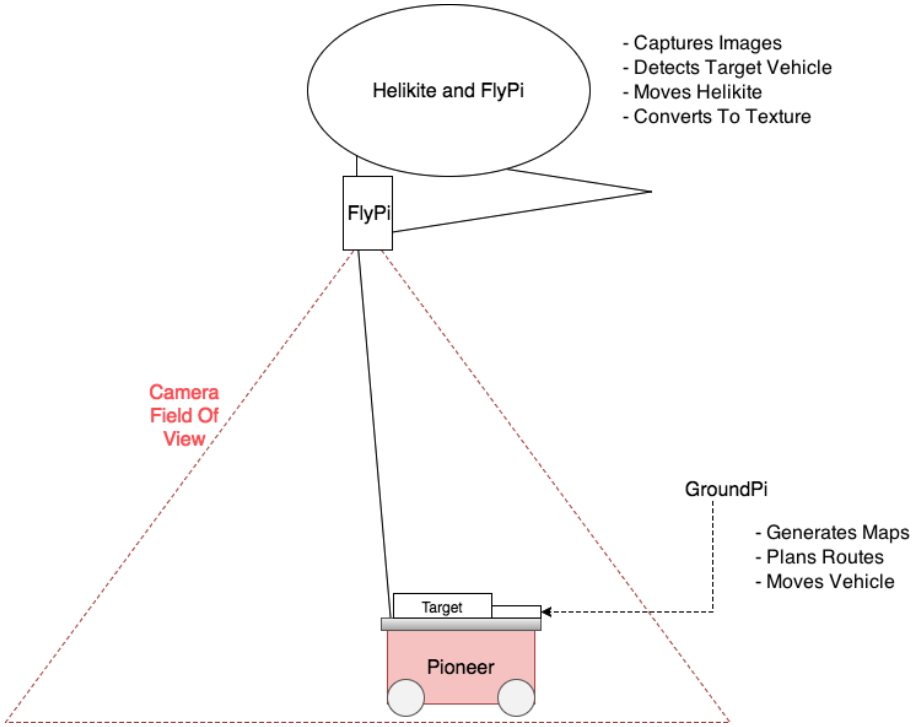


Figure 3.2: System overview

Table 3.1: FlyPi hardware

Camera	**Controller**	**Motor**	**ESC**	**Propeller**	**Power source**	**Controller-backup**	**Motor-Backup**
Raspberry Pi Camera	Raspberry Pi B V3	Emax CF2822 Brushless	Hobbyking HK20a Brushless	GWS EP-9047	Tether	2x3.7v 155mah Li-Po	2x3.7v 155mah Li-Po

Figure 3.3: Pioneer robot

3.5 Software components

The software distributed over the FlyPi and GroundPi components performs multiple tasks concurrently and in real time. Table 3.2 details the software components and whether it would be feasible to perform the task on the FlyPi, GroundPi or both.

The goal when deciding where to perform each piece of processing was to balance distribution, use minimal communication and allow for redundancy. The requirements were somewhat intertwined. Given the high cost of transmission, it was decided to transmit from the FlyPi and GroundPi after turning the captured imagery into a texture based image, therefore

Table 3.2: Task assignment

Task	**Possible task location(s)**	**Chosen location**
Image Capture	FlyPi	FlyPi
Vehicle Detection	FlyPi / GroundPi	FlyPi
FlyPi Motor Control	FlyPi	FlyPi
Image Stitching and Texture Image Generation	FlyPi / GroundPi	FlyPi
Transmission / Receipt	Both Required	FlyPi/GroundPi
Mapping	FlyPi / GroundPi	GroundPi
Route-Planning	FlyPi / GroundPi	GroundPi
Vehicle Movement	GroundPi	GroundPi

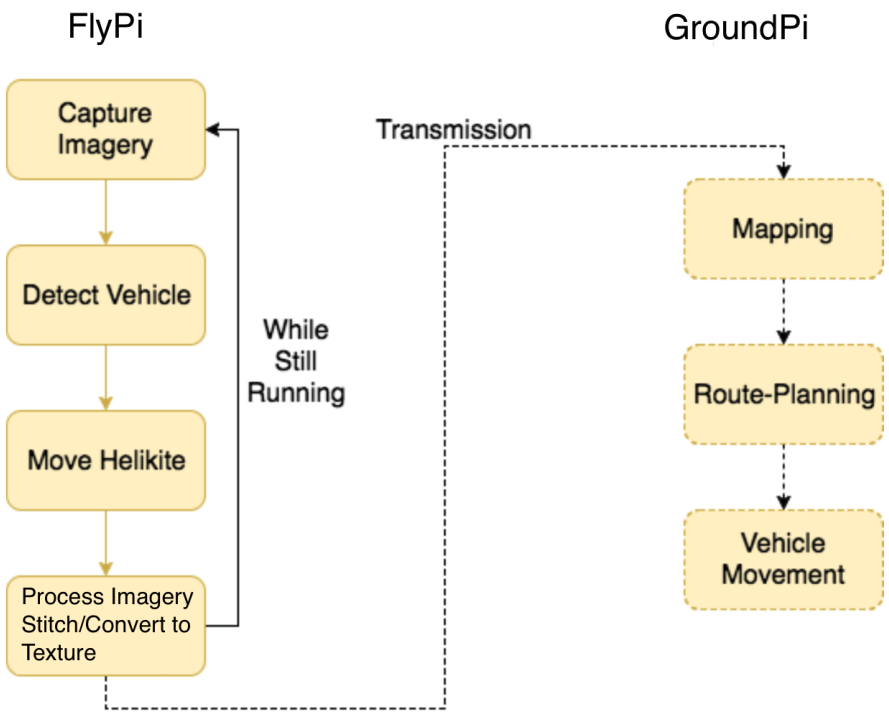


Figure 3.4: Software structure

reducing the image size and minimising communication. Sending the image after texture image generation also had the effect of balancing the distribution requirement in that each component is not waiting excessively long amounts of time for the other half to complete its task. Temporary redundancy was maintained in the current system with the FlyPi being able to move and follow the GroundPi without connection and the GroundPi storing the map and performing movement.

The layout of the final system is shown in Figure 3.4.

3.5.1 FlyPi

Image capture: Images are captured from the Raspberry Pi camera at 2592×1944 pixel resolution using the OpenCV V4L2 driver.

Vehicle detection: Pattern-based roundel detection using the ratio of black to white pixels in vertical horizontal sequences is used to detect pre-defined roundel patterns. Roundels are then grouped into a pre-defined triangular target configuration using their relative sizes and locations.

FlyPi motor control: After vehicle detection, the physical distance between the ground vehicle and the centre of the captured frame are used to generate appropriate motor commands.

Commands are generated using a proportional P algorithm with alterations made to allow for the flight characteristics of the Helikite platform including getting lost ahead or behind the target. Motor control commands for the ESC are generated using pulse width modulation.

Image stitching and texture image generation: After capture and detection, each image is masked to remove the robot. The ORB feature detector is used on the masked image to detect feature points. Between consecutive frames, the detected features are then matched with erroneous matches removed. The resultant matches are subsequently used to generate homographic transformations between frames, as well as to stitch frames together before transmission. Upon receiving an update request from the GroundPi, the stored transformations are used to generate a sliding window stitched image of 2 frames and locations relative to the original transmitted frame. After generation of stitched images, these are converted to a texture based image using a speeded-up GLCM and a normalised combination of correlation, contrast and entropy before transmission to the GroundPi.

Transmission: Texture-based images and their coordinate details are transmitted to the GroundPi using the boost library for addition to the map after receipt of an update request from the GroundPi.

3.5.2 GroundPi

The GroundPi receives input in the form of texture based, stitched images from the FlyPi and completes the following software tasks:

Texture based similarity mapping: Upon receipt of the first image, the mapper generates an empty map using the MRPT (Mobile robot planning toolkit) library and based on the size of the desired path. The area around the vehicle at the start of the image is used as the basis for texture-similarity based map updating using the deviation from this as a value of safety using updates from the FlyPi.

Route planning: Given a start and goal location, an initial path is planned using an RRT-based planner and path smoothing as a series of waypoints. During execution, the path is either fully or partially replanned if subsequent updates cause a blockage during execution. The decision whether to fully or partially replan depends on the percentage of waypoints blocked.

Vehicle movement: Given a path from the planner as a series of waypoints the path is executed using a rotate and move forward strategy. During execution, locational updates are provided to the GroundPi from the FlyPi. During execution, as waypoints are reached, they are removed from the path until the final goal is reached.

Table 3.3: Field of view and pixel size examples

Height above ground (M)	Horizontal F.O.V (M)	Vertical F.O.V (M)	Horizontal pixel size (CM)	Vertical pixel size (CM)
1	1.00	0.74	0.08	0.08
2	1.99	1.47	0.15	0.15
3	2.99	2.21	0.23	0.23
4	3.99	2.94	0.31	0.30
5	4.99	3.68	0.38	0.38
6	5.98	4.42	0.46	0.45
7	6.98	5.15	0.54	0.53
8	7.98	5.89	0.62	0.61
9	8.97	6.62	0.69	0.68
10	9.97	7.36	0.77	0.76

3.6 Flying height and field of view

The selection of an appropriate flying height for the aerial vehicle was dependent on a variety of factors. For the indoor testing, the height of the aerial vehicle was constrained by the ceiling height, whereas for the outdoor tests this was limited only by the maximum length and weight of the tether and the requirement for an effective capture resolution and field of view.

The chosen hardware (Raspberry Pi camera V1) has a horizontal and vertical field of view of 53.5 and 41.41 degrees respectively. Using the horizontal and vertical fields of view and trigonometry as shown in Figure 3.5, the field of view can be calculated at 1.00m horizontal and 0.74m vertical for every metre in height above the ground of the aerial vehicle.

Before stitching and transmission, the images are captured at 2592×1944 pixels and reduced to 1248×792 pixels in order to increase speed, therefore the real world size for each pixel corresponds to the real-world field of view in each plane divided by the number of pixels, some sample heights are shown in Table 3.3.

For the interior tests, the height was approximately 4m which corresponds to a $3.99 \times 2.94m$ field of view, and average pixel size of 3.8mm before transmission. Because of the uncontrolled direction of the Helikite and the fact that it was being towed, the height remained fairly constant but the location of the ground vehicle within the frame was not. The height of the aerial vehicle during outside capture is approximately 7m, which gives an increased field of view ($7.98 \times 5.15m$) at the expense of a decrease in real-world pixel size. The increased field of view provided by the outdoor tests may be useful in order to offset the less-constrained behaviour of the Helikite in the outdoor environment.

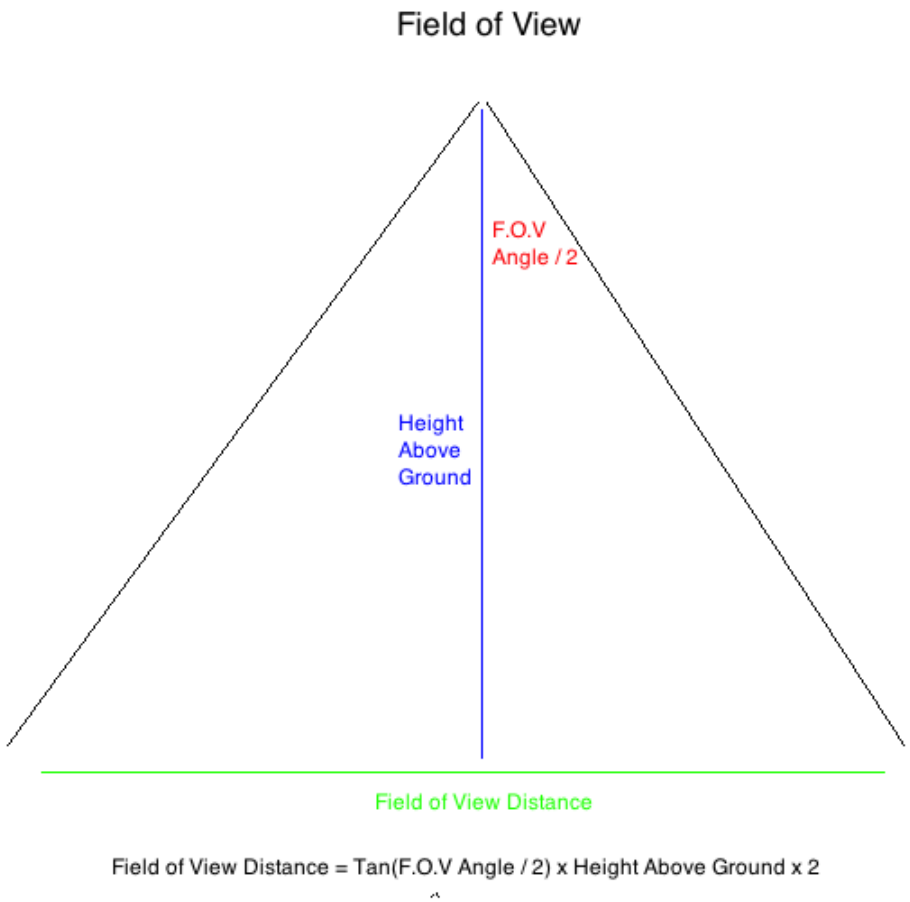


Figure 3.5: Field of view calculation

Chapter 4

Hardware selection

4.1 Introduction

The hardware requirement for this project was for an aerial vehicle to capture overhead imagery in real time covering the space surrounding an unspecified ground-vehicle. It must be able to stay flying for extended periods, capture high-resolution imagery, operate in a variety of weather conditions, be stable and controllable.

Despite the current trend towards using quadcopters, the eventual design consisted of the bespoke HARPP system (Helikite Aerial Powered Platform) modelled on the Allsopp Helikite with a tethered custom-made control/power unit (FlyPi). The inclusion of a motor allows experimentation with powered Helikite flight. Due to the selection of bespoke flying hardware, a ground unit (GroundPi) was also produced to enable wireless communication between flying and ground vehicle in addition to real-time path planning. This makes the use of different ground vehicles possible with minimal alteration to the core system.

This section explains the structure of our hardware implementation and the design choices that lead to its adoption.

4.2 Aerial platform selection

The aerial platform must support an image camera and transmission system, be able to follow the ground vehicle, be deployable (transportable using a van or boat) and display sufficient endurance to allow practical usage in often inaccessible environments. Available off the shelf products include quadcopter-based platforms such as the DJI Phantom[1], KAP (kite aerial pho-

[1] Dji, "DJI Phantom Drone | DJI." url: http://www.dji.com/product/phantom, 2015 (Accessed on 05/02/2016).

Table 4.1: Aerial platform characteristics

	Deployability (physical size)	Endurance	Versatility	Stability	Controllability
Kite	Good (Small)	Dependent on wind-conditions	Requires specific wind conditions	Dependent on variability of wind conditions	Variable
Helicopter	Good (Small)	Minutes	Requires low winds	Unstable (Requires Constant Control Inputs)	Good
Quadcopter	Good (Small)	Minutes	Requires low winds	Unstable (Requires Constant Control Inputs)	Good
Plane	Variable	Minutes-Hours	Requires low winds	Stable in normal forward flight	Good
Blimp	Variable	Hours	Requires low winds	Stable	Medium (Dependent on wind levels)
Helikite	Variable	Hours	Good (Accepts low to medium wind-conditions)	Stable	Unknown

tography) platforms[2] [115], gas-powered professional helicopters or toy blimps. Table 4.1 shows the properties of evaluated systems. Bespoke hardware platforms were also evaluated including kites, helicopters, blimps and planes. Previous work at this institution had used kites [115], but these were discounted due to stability, control and deployability issues.

The platform chosen was an Allsopp Helikite[3] (see Figure 4.1) due to its stability and excellent lift. Helikites, similar to blimps/aerostats are non-rigid gas-filled balloons that gain lift from a helium filling. In contrast to blimps, the balloon is attached to a kite-like wing which produces additional lift in windy conditions, maintains stability and causes the vehicle to tend to face into the wind, making the movement more predictable. There was little information available on powered Helikite flight, its other properties were sufficiently positive for us to try it anyway and describes the first real-time online HAPP.

HAPP (Helikite Aerial Photography Platform) literature has found Helikites to be deployable, portable, and have a minimal ongoing cost [105, 107] though it has been noted that when unpowered they are difficult to accurately position. This project aims to rectify this with a

[2]KAPshop, "KAP Starter Kits - KAPshop." url: http://www.kapshop.com/Starter-Kits/c86/index.html/, 2015 (Accessed on 05/02/2016).

[3]Allsopp H. Limited. Allsopp Helikites and Accessories / Support Products. url: http://www.allsopp.co.uk/index.php?mod=page&id_pag=5, 2015. (Accessed on 29/03/2017).

Figure 4.1: Helikite platform in flight

propeller-based solution.

When selecting the size of Helikite compromise was made between the deployability of the platform, the quality of imagery and the longevity of flight. The quality of imagery is proportional to the size of Helikite with larger sizes having a greater maximum payload of better quality image capture equipment. Because of the size and deployability limitations, positive experiences in prior literature, research novelty and its fulfilment of the requirements a 1.6m^3 Allsopp Skyhook (Table 4.2) was chosen as the platform for this work.

4.3 Aerial mass budget

This project required a compromise between mass and functionality, such as camera quality for the FlyPi components with a preference for reducing mass.

The selection of a 1.6m^3 Allsopp Helikite resulted in a payload mass limit of 300g. Although the design of the Helikite allows for lift of up to 3kg in a 24kph wind, the mass was

Table 4.2: Helikite properties

Type	Skyhook
Capacity	1.6 m^3
Thickness	0.05mm
Lift in no wind	300g
Lift in 24kph wind	3kg
Max wind speed	48kph
Max altitude	750m
Length	180cm
Width	135cm
Cost ex. VAT	£550

limited to enable flight in little or no wind. This subsection describes the initial choices behind the selection of the FlyPi hardware.

4.3.1 Camera/control components

Various lightweight camera systems were evaluated in order to find a suitable capture system. GoPro action cameras, inexpensive analogue cameras, webcams and camera module boards were evaluated. Typically they either produced low-resolution imagery or would be complex to transmit/integrate live feeds with the rest of the system (see Table 4.4).

Previous literature has not attempted to power the Helikite platform but suggests that positioning can be difficult. In [105], the author notes that "Walking the Helikite around and allowing it to ascend or descend to various altitudes is the only way this aircraft can be moved into position, it remains rather challenging to accurately establish a precise camera location". The controllability difficulties lead to the decision to attempt to power the vehicle, however, this subsequently placed more demands on both the mass and power budget. The motor controller requirement and the need for a transmitted camera feed meant that the options were to either:

1. Run the camera separately from the motor controller.

2. Combine the two systems and potentially save mass.

Although option 1 (running the camera and motor controller separately) would have been easier to implement, this configuration introduces latency into the control loop. A Raspberry Pi was selected as the controller as it is able to capture, transmit, control a motor using PWM and had compatibility with OpenCV in a small form-factor. Because of its low-weight (~3g),

Table 4.3: FlyPi hardware

Camera	Raspberry Pi Camera
Controller	Raspberry Pi B
Motor	Emax CF2822 Brushless
ESC	Hobbyking HK20a Brushless
Propeller	GWS EP-9047
Power source	Tether
Controller backup	2x3.7v 155mah Li-Po
Motor backup	2x3.7v 155mah Li-Po
Mass	185g

high resolution, availability and ease of integration with the Raspberry Pi, the Raspberry Pi Camera Module was chosen to accompany the Raspberry Pi.

A brushless motor and controller were selected. Thrust-tests were performed with suitable motors and different propellor combinations, the candidate motor was then tested on the Helikite to test its suitability which proved successful. Note that the 300-gram mass budget includes the power system which comprises onboard batteries (backup) and power tether.

The final components are detailed in Table 4.3. The electronic components were combined onto a battery-powered system (See Figure 4.2) using tethered power. This combines and integrates the capture and power system with appropriate mounting points for the Helikite. Since the initial build of the FlyPi board, the only physical change has been the upgrading of the Raspberry Pi to a newer more powerful model with the associated increase in processing speed.

4.4 Power system

Initial flights concentrated on learning the unpowered flight characteristics of the Helikite and as such only captured imagery. These used an off-the-shelf USB backup battery (commonly used for mobile phones) and gave a capture time of ~1-2 hours. After evaluating the captured imagery, it was clear that whilst under tow the Helikite was trailing behind. Once the decision to attempt to power the Helikite was made, a prototype control board was built to hold a motor, camera and power supply, and in addition to the USB battery, a second Li-Po was provided to power the motor and speed controller. The particular motor was selected based on its perceived suitability in terms of thrust, current draw and weight. The prototype FlyPi had its power sources separated to avoid interference and to limit the potential for high current draw from the motor resetting the capture system in-flight.

Initial testing of the power system varied, bench-tests at various speeds, with and without

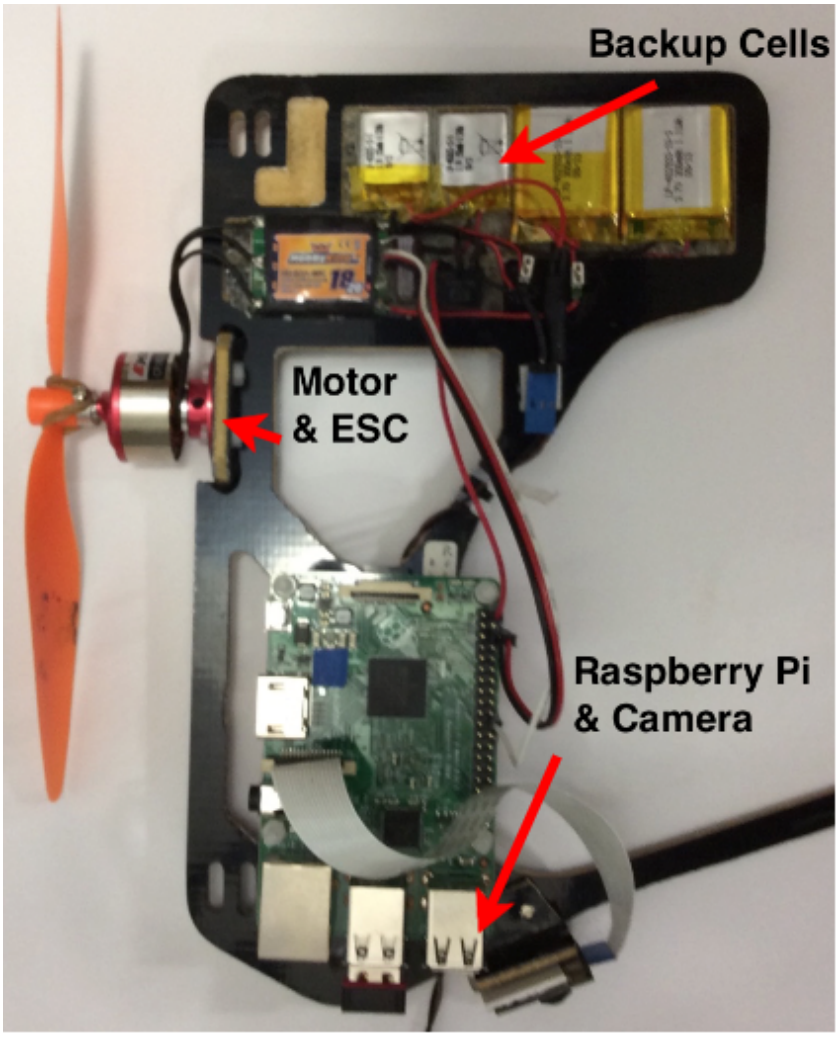


Figure 4.2: FlyPi

Table 4.4: Comparison of camera types

Camera Type	**Example**	**Example mass**	**Example power supply**	**Max resolution**	**Colour**	**Inbuilt transmission**	**Cost**
Action Camera	GoPro Hero+	156g	Self Contained Li-Po	1920×1080 pixels	Full-Colour	Yes	£169.95
Mobile Telephone	Motorola moto g	155g	Self Contained Li-Po	1280×720 pixels	Full-Colour	Yes	£69.99
Consumer Digital Camera	Vivitar S126 Compact Digital Camera	100g	$3 \times AAA$	16 megapixel	Full-Colour	No	£19.99
CMOS Camera Sensor	CMOS Camera Module (Sparkfun)	26g + attached controller	Powered by attached controller	728×488 pixels	Full-Colour	No	~£22
Analogue CCTV-Type Camera	Wireless spy camera (eBay)	Unavailable	Requires 8v Supply	628×582 pixels	Full-Colour	Yes	£43.90
Raspberry Pi Camera Module	Raspberry Pi Camera Module	3g + Raspberry Pi	Powered by Raspberry Pi	2592×1944 pixels	Full-Colour	No	£13

different propellers were completed to ascertain current draw and thrust. After initial bench-tests, static outdoor tests were used to gain a rough idea of power versus speed, performance in wind and flight behaviour of the motor/power unit. Performance of the prototype varied, whilst the camera and transmission system were adequately powered for long periods, the motor would show a decrease after a few minutes and shut down shortly after, primarily due to the limited size of cells being carried (due to weight) quickly becoming depleted. The majority of platforms in the literature use a non-fixed power supply i.e. batteries or fuel, however, there has been some use of tethered power systems where longevity is required. To evaluate tethered power, we began by with the current draw of the motor, ESC and capture system chosen (Table 4.5), we then looked at various lengths and types of tethered cable to measure loss over distance, mass and anecdotally evaluate the suitability of different cable options. The cable chosen was a twisted bell-cable (Figure 4.3) selected in order to keep weight to a minimum whilst allowing delivery of sufficient power. In comparison to a battery system delivering the same amount of power, the tether is significantly lighter at low altitudes. At higher altitudes, depending on the alternative battery chosen, the battery system will be lighter (where the mass of the wire is greater than a battery cell), however, longevity is likely to be severely limited.

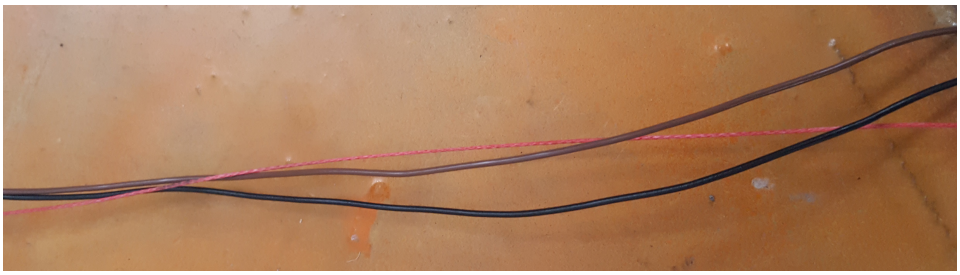


Figure 4.3: Line and power tether

Table 4.5: FlyPi power requirements

Component	Max Current Draw
Raspberry Pi + Camera	~500mA
Brushless Motor	~18A
Brushless E.S.C	Unknown

In subsequent indoor and outdoor longevity testing, the powered tether performed acceptably both in terms of a reduced payload and increased longevity. However, it was noted that there were occasional problems with a purely tethered system which included a "drop" of power and subsequent re-setting during prolonged motor-runs. The final power solution is a hybrid system, main power is supplied over the tether but small cells are provided on the control board to normalize fluctuations in current and allow for momentary losses of power.

Whilst battery-based power systems are common in previous works, there is some evidence of tethered power/control systems in the literature [37, 42, 55, 59]. Some of the quadcopter-based solutions have used an auto-reeling system in order to maintain tension in the tether. As well as endurance, the use of a tether can help to stabilize the platform (particularly in gusty conditions) reducing the task-space [37, 55] and whilst the other works use an always-connected tether, Garone et al [55] hypothesizes the potential for an as-needed connection, though in practice the practical implementation has yet to be tested.

Battery-based solutions have the advantage that they do not require ground power infrastructure, may be more easily deployed and that flying height is uncoupled from the mass budget. However, batteries are limited in longevity and require downtime between charges. Tether-based solutions are typically used where longevity is required and may be more cost-effective but are more difficult to deploy, may affect the flight characteristics of the vehicle and introduce coupling between altitude and mass.

Table 4.6: Comparison of board materials

Material	**Example thickness**	**Example mass** (10×10**cm**)	**Rigidity**	**Strength**
Plywood	5mm	23-26 grams	Good	Good
Plastic (Perspex Sheeting)	2mm	~20 grams	Acceptable-Good	Good
Glass-Fibre	2mm	42 grams	Excellent	Good
Carbon-Fibre	2mm	32 grams	Excellent	Excellent
Aluminium	1.2mm	32 grams	Good	Excellent
Carbon-Foam Sandwich	5mm	52 grams	Excellent	Excellent

4.5 Control board material / Helikite alterations

The control board was designed to be fixed to the Helikite using the pre-existing Velcro fasteners to allow removal without damage and minimal alterations to the Helikite itself.

The prototype FlyPi was built from plywood and intended to evaluate flight and capture performance whilst experimenting with different mounting points on the Helikite and for the camera. The subsequent final board was modified in that it has an adjustable mount as well as strategically placed holes and recesses to reduce weight and protect sensitive components. When selecting the board material, wood, plastic and composites were evaluated (Table 4.6) ultimately leading to the selection of the carbon-foam sandwich material.

After the motor selection, various alterations were made to the Helikite for safety purposes. The stock Helikite's kite has a vertical component on which we mounted the control board, initially this was unsupported but tests showed that this was liable to movement in wind, potentially damaging the balloon with the propeller. The Helikite was modified to add a vertical strut along the kite to hold it in place and reduce the risk of damage. In order to save mass, the vertical strut was constructed from carbon fibre tubing and an aluminium collar and allowed the removal of an existing horizontal carbon-fibre strut. The similar masses of the additional and removed strut resulted in a negligible difference in mass before and after modification.

4.6 Ground system

Direct FlyPi to vehicle communication was considered, with planning and mapping performed on the robot. However, the requirement for ground robot variability introduced the requirement to install software on multiple vehicles with different software and hardware requirements and variable processing powers. Ultimately it was decided to use a ground robot in-

Figure 4.4: GroundPi

terface (GroundPi) to provide a standard interface between the flying vehicle and the ground robot whilst sharing the computational load.

Various options for interface computer were considered and included netbooks, laptops and single-board computers but ultimately a Raspberry Pi was selected. The Raspberry Pi was selected because of its known compatibility with the pre-existing FlyPi controller, compatibility with many of the required libraries, low mass and low power requirements. Communication was an important factor with the Raspberry Pi allowing simultaneous wireless and wired communication to the ground and flying vehicles.

The completed GroundPi hardware is shown in Figure 4.4 and consists of a Raspberry Pi and long-range antenna. Communication between the FlyPi and GroundPi uses a wireless LAN whereas the communication between the GroundPi and the ground vehicle uses wired Ethernet.

4.7 Computational considerations

4.7.1 Balancing computational power versus mass

The platform is distributed and although some tasks could be performed on either the GroundPi or FlyPi some such as image capture or robot interfacing are location-specific. Tasks that are performed on the FlyPi are subject to its limitations in processing power. FlyPi in its most basic form must capture imagery and transmit imagery. This implementation also incorporated motor-control, vehicle detection and basic image processing into the FlyPi because of the capacity of selected hardware (Raspberry Pi) to do these tasks.

Micro-controllers were quickly discounted, although options exist such as the Arducam + Bluetooth adapter [4], the additional shields and modules add weight, have a low transmissions speed and lack the option to perform image preprocessing before transmission. Mobile telephones were ideally suited due to their longevity, cost, connectivity and image quality but were disregarded due to the requirement for motor control.

It was quite obvious that a single-board computer was the most viable solution. At the time of building, the Intel Galileo platform supported Linux but was expensive, heavier and anecdotally slower [5]. Beaglebones similarly run Linux but have poorer performance than the Raspberry Pi [6]. All of the platforms have GPIO pins. The Raspberry Pi was ultimately chosen because of its compatibility with Linux and OpenCV, availability of a high-resolution camera module, and its superior theoretical performance in comparison with the Beaglebone and Intel Galileo.

4.7.2 Communication methods and hardware

The distributed system requires data transmission between FlyPi and GroundPi. Data transmitted between platforms consists of imagery as well as positional information and should be transmitted quickly and without loss. When selecting a communications method, the speed, reliability, ease of integration and hardware weight were evaluated.

A wired solution had the advantage of reliability and speed but would undoubtedly affect the Helikite performance due to increased mass and tethering and was therefore disregarded. Optical methods such as Infra-Red are low-bandwidth and unreliable. Wireless methods have

[4] Lee Jackson. Arducam + Bluetooth Module. url: http://www.arducam.com/arducam-bluetooth-module-wireless-image-system/ (Accessed on 02/16/2016).

[5] Intel Galileo Review | Linux User & Developer - the Linux and FOSS mag for a GNU generation. url: http://www.linuxuser.co.uk/reviews/intel-galileo-review. (Accessed on 02/16/2016).

[6] Raspberry Pi 2 vs. B+ & Beaglebone | Initial State. url: http://blog.initialstate.com/pi-2-vs-b-vs-beaglebone/. (Accessed on 02/16/2016).

the advantage of potential speed, reliability and limited additional weight requirements, Bluetooth is short range but has a low bandwidth so was disregarded. The final selection was a Wifi-based network which is fast (depending on the wireless protocol used), has high reliability, is easily integrated through pre-existing libraries/protocols and no additional weight (using the onboard wifi chip) that is not coupled to altitude.

After selecting the Wifi-based communication method, It was decided to use the Boost [7] library to serialize the data from OpenCV as well as managing the transmission over IP of the image and positional data.

4.7.3 Distributing computational tasks

Distribution of different computational tasks over the course of this project has varied as the hardware configuration has changed but in general, has followed several concepts:

- Load Balancing: Computational load should be balanced evenly between Aerial and Ground platforms such that neither is required to wait significant periods to "catch up".
- Minimal Communication: Since communication is costly in terms of time and potentially subject to interference, communication should be kept to a minimum.
- Redundancy: There should be protection against errors such as disconnections and procedures in place to re-establishing normal behaviour.
- Abstraction: The ground vehicle should be variable.

There are a variety of tasks that the distributed system must perform (Table 3.2), the aim being to conform to all of the above concepts. As the communication is one-way the distribution consists of a virtual line (transmission) placed in the sequence separating the FlyPi and GroundPi tasks, the upshot is that if the transmission is incorrectly timed, it may require bi-directional transmission or may be imbalanced.

If the transmission took place immediately after image capture (requiring the FlyPi camera) but before the motor-control, the problem would become bi-directional requiring transmission of location or motor commands to the FlyPi, therefore, increasing the potential for errors. Alternatively, transmitting too late risks imbalance between the components and can cause bottlenecks or lack of redundancy for the GroundPi.

The final system transmits the image after capture, detection, motor control and image processing thus completing the required tasks on the FlyPi without transmission, maintaining

[7] Rachel Gregory. Boost C++ Libraries. url: http://www.boost.org/, March 2013. (Accessed on 02/16/2016).

singular transmission and balance between hardware. After transmission, the image is integrated into the map and then used to route-plan before talking to the vehicle, this facilitates redundancy and allows the ground-vehicle to function without new information if a connection error occurs.

Chapter 5

FlyPi

5.1 Introduction

The flying controller (FlyPi) keeps the target vehicle and its controller (GroundPi) in view whilst providing imagery and positional information in real-time to the GroundPi unit. To remain above the target vehicle the offset between the desired and current location is identified and appropriate movements are made using an altered P (proportional) controller and Euclidean distance measure. Prior to transmission images are transformed relative to the original base image using ORB feature detection with brute-force matching and mis-stitch detection and then stitched. Stitched images are subsequently converted to greyscale texture intensity images using Haralick's texture features [62] and a combination of correlation, contrast and entropy with speed improvements in order to run in real time. This section explains the software design choices taken as part of the FlyPi controller with particular emphasis on real-time usage. The basic software structure is shown in Figure 5.1.

5.2 Keeping above the target

To provide a suitable field of view, the target vehicle must be kept in the frame as much as possible whilst maintaining an acceptable frame rate. A combination of target detection and an altered P (proportional) controller was used with PWM (pulse width modulation) to control the FlyPi motor, attached to the Helikite.

5.2.1 Detecting the target

Object tracking was initially considered (i.e. finding and following the vehicle between frames based upon its known parameters) but was discounted because of the erratic movement of both

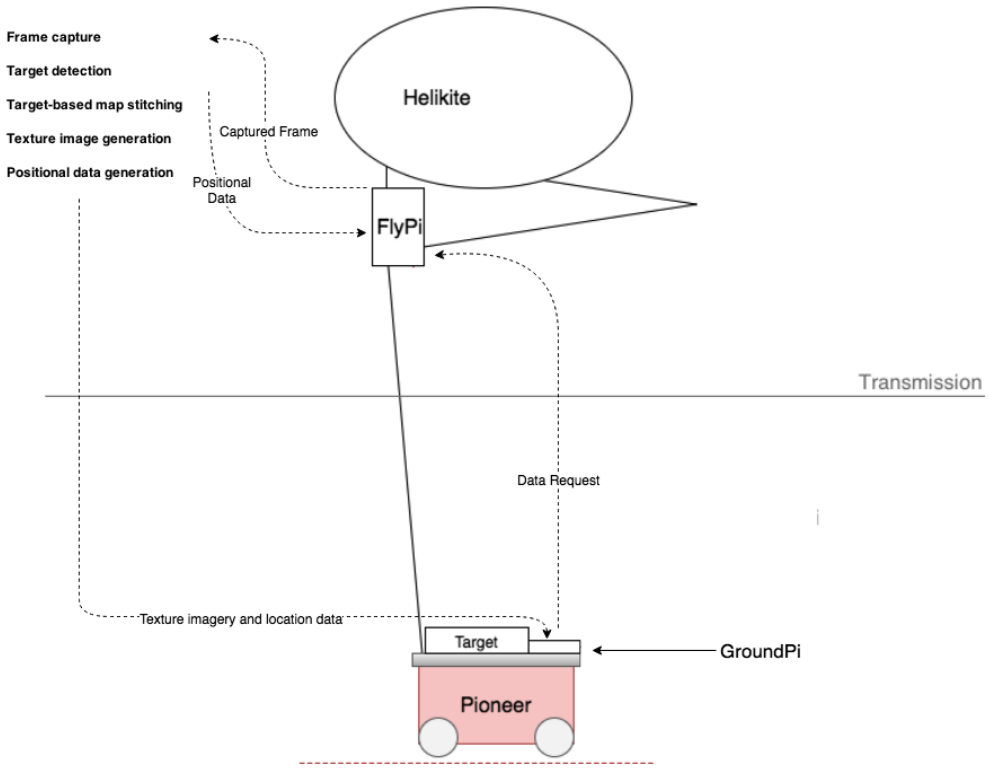


Figure 5.1: System overview

the FlyPi and the camera, the potential for loss and subsequent reacquisition, the possibility of using different vehicles and the low-powered hardware. The relative motion between the ground vehicle and the flying vehicle can be erratic and unpredictable. Flying vehicle movement is largely dependent on wind speed, direction and tow (possibly including changes in rotation and height). Ground vehicle movement is based on the path-planned with neither vehicle necessarily being stationary between frames. Ultimately, detection in each frame was used in the almost certain knowledge that each frame would contain a single instance of the target which could then be used to ascertain location as well as scale. Whilst it would have been possible to alter the robot to form a target of sorts (for example by painting it luminous orange to detect via the HSV colour-space), this was considered impractical so an adhesive target was used instead to facilitate the use of multiple vehicles and simplify the detection process.

Various real-time target detection methods were evaluated over the course of the project with a requirement for speed, accuracy, resilience and scale. The chosen method of target detection used by the system uses prior unpublished work by Neal Snooke because of its simplicity and speed using an isosceles roundel configuration as the target (See Figure 5.2a). This work uses pattern-based matching of roundels (matching sequences of black and white pixels from binary images given known parameters) shown in Figure 5.2b. First searched (and matched) horizontally, the pixels are then searched vertically to verify the potential roundels, the resultant roundels are then grouped in order to obtain a three roundel target. This prioritises orientation which would not be obtainable with a single circular target alone.

The original method works indoors under uniform illumination and is computationally efficient, but suffered various problems such as mis-thresholding, occlusion, variable illumination, erosion of roundels and incomplete detections when deployed outside on this hardware. These problems prevented successful detection in the majority of cases. Various other methods were evaluated including Hough circles and detections of contours but were too complex to run in real time (even with a reduced search space) on the Raspberry Pi V2. Ultimately it was decided to use this roundel-based method as the basis for the low-cost target detection method used in this work.

After finding candidate roundels, the roundel detector (described in Algorithm 1) verifies the conformity of candidate roundels to the ideal roundel model, this is run horizontally and then the candidate locations are checked similarly vertically with candidates stored as Target objects if they conform to certain roundel parameters either vertically or horizontally and are of appropriate size (using Algorithm 2), then grouped by location and relative size. Selection hypothesizes that larger roundel groups are more accurate but smaller groups are sufficient to give approximate location, where multiple groups are present, candidates are selected by size,

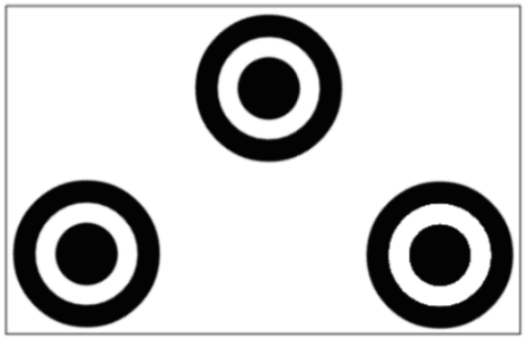
conformity to ideal roundel/target parameters (and size in the case of single roundel groups). This work differs from the original detection algorithm by using a more complex set of roundel detection, grouping and selection parameters (to account for deviation from ideal roundels in imagery) in order to efficiently identify targets either fully or partially in frame. Modifications included:

- Mis-thresholding/variable illumination: The binary thresholder was modified to more successfully threshold the target imagery in varying illumination conditions.
- Occlusion/incomplete detections: The grouper was modified such that it could accept pairs or even singular roundels in place of the full target, albeit with a decrease in accuracy.
- Erosion of roundels: The roundel detector was modified such that it could accept the erosion of the roundel, usually characterised by thinning of the black rings in the roundel caused in brighter lighting.

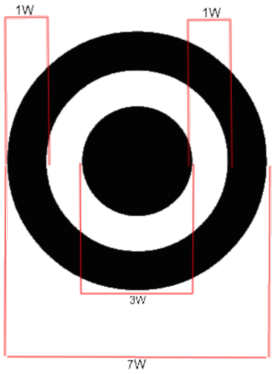
Subsequent hardware revisions have made feasible (given the real-time constraint) different algorithmic possibilities (such as a canny-based glyph detector that would not have been possible with earlier hardware). However, these hardware improvements have also meant the low-cost roundel-detector now works faster but was replaced in some indoor tests with a canny-based glyph detector where this was more efficient (particularly in extremely bright environments where the roundels tend to be "washed out") based on an online tutorial by Ross D Milligan[1]. The glyph detector as implemented uses Canny based contour detection, followed quadrilateral grouping and pattern matching on the glyph to confirm the correct glyph pattern has been detected.

In its simplest form, the roundel detector will evaluate every single pixel in the target image on the findhorizontalTargets method, subsequently grouping them and adding them to a list of candidate targets, the roundels (a subset of the image) will then be checked vertically in order to generate a list of candidate roundels for grouping. The evaluation of the pixels in the first step in its best and worst cases is of the order O(n) where n is the sum of pixels in the input image as every pixel will be evaluated. In the second step (vertical checking), the theoretical minimum number of pixels evaluated is equal to zero if there are no candidates or every pixel if there are candidate roundels filling the entire image, which again evaluates to O(n), resulting in an overall best and worst complexity of O(n) for the roundel detection. By comparison, the

[1]Ross D. Milligan. Glyph recognition using OpenCV and Python, July 2015, URL: https://rdmilligan.wordpress.com/2015/07/19/glyph-recognition-using-opencv-and-python/ (Accessed 12/10/2016).



(a) Target design



(b) Roundel schematic

(c) Example image

Figure 5.2: Roundel detection components

Canny edge detector as used in the glyph detector alone has a complexity of O(n log n) before sorting, polygon generation and sequence matching.

5.2.2 Moving the Helikite

Prior work using Helikites primarily concentrated on using them as a fixed or towed platform without an independent propulsion system and as such, the flight parameters were relatively undocumented. The Helikite itself has the advantage that it has a certain amount of inherent lift (due to the helium filling) with the additional lift being generated by the kite attachment.

The Helikite exhibits varying flight characteristics depending on the current wind speed and direction, relative towing vehicle location and speed (see Figure 5.6). In no wind and without vehicle movement the Helikite will be positioned directly above the target in an arbitrary direction. When the wind is present but the vehicle is not moving the Helikite will be positioned above the vehicle facing towards the wind (with additional lift). Most importantly if the Helikite is not above the target vehicle (due to towing), it will usually face towards the towing vehicle and be behind it in the direction of travel, therefore applying the correct amount of power should bring it above the target vehicle.

In order to generate appropriate movement commands, the limitations and flight characteristics were taken into consideration, the goal is to keep the robotic vehicle as close as possible to the centre of the frame in order to maximise the usefulness of the captured imagery (showing the area surrounding the vehicle). Because the Helikite will either be located above or facing towards the target vehicle, motor-power commands using physical distance (based on target scale and Euclidean pixel distance) can be used to move the Helikite as close to the target as possible.

As well as the target location, images are also required to build a world map using target location, image stitching and texture-based segmentation.

5.3 Locating the vehicle and stitching imagery

Stitched images comprised of two frames converted into a texture-similarity map are transmitted in conjunction with locations relative to the base (start) image. Two frames were selected to provide a sufficient number of updates and to reduce the amount of transmission. Images are initially stitched and transformed using an ORB[99] based detector and brute-force matcher, erroneous matches are then removed using Lowe's ratio and other forms of error-checking.

Once matches are identified they are stored to be used for later stitching and to obtain relative location.

5.3.1 Detecting features

A variety of feature detectors were evaluated (those implemented in the OpenCV library for simplicity of implementation) with the requirement that they operate in real-time, are robust and can detect features in a variety of types of scene. Due to the type of camera movement, the chosen planner must also be invariant to unpredictable rotations and changes in scale.

SIFT [94] was primarily discounted because of its speed (not being suitable for real-time usage with images of this size but scale and rotation invariant). SURF [98] though speeded up from SIFT uses proprietary code. FAST [92, 93] is a high-speed corner detector that is suitable for SLAM applications but is not resilient to noise.

ORB [99] is a fusion of the FAST detector and BRIEF [100] descriptor with performance improvements, has been shown to be faster than SIFT/SURF and so is more suited to low-power applications (such as this). ORB was ultimately chosen because of its speed, robustness with changes to scale/rotation and non-proprietary licensing.

The ORB detector accepts a single parameter (maximum number of features) which was set to 750, tests showed that despite a negligible improvement in terms of speed by using a lower number, these matches were often not of sufficient quality and particularly in sparsely populated scenes tended to not be distributed across the image (causing poor stitching). All of the feature detection work was carried out using the OpenCV library [2].

5.3.2 Matching features

Once consecutive images have had features detected, these are matched using a Brute-force matcher, erroneous matches are then removed using a combination of outlier removal and Lowe's ratio test.

OpenCV's matcher class provides two different types of matchers (Brute-force and FLANN). Brute-force simply finds the best match (or N matches) purely in terms of some distance measurement, whereas FLANN (Fast Library for Approximate Nearest Neighbors) uses multiple algorithms optimised for large datasets and is typically faster than brute-force when using such large sets of data. Since the dataset used herein is comparatively small (<1000 points per image) the brute-force matcher was chosen with Hamming distance as its distance comparator and cross-checking (confirming the match is best in both directions) disabled in favour of Lowe's ratio test.

[2] OpenCV Team. OpenCV library. URL http://www.opencv.org/ (Accessed 20/12/2017)

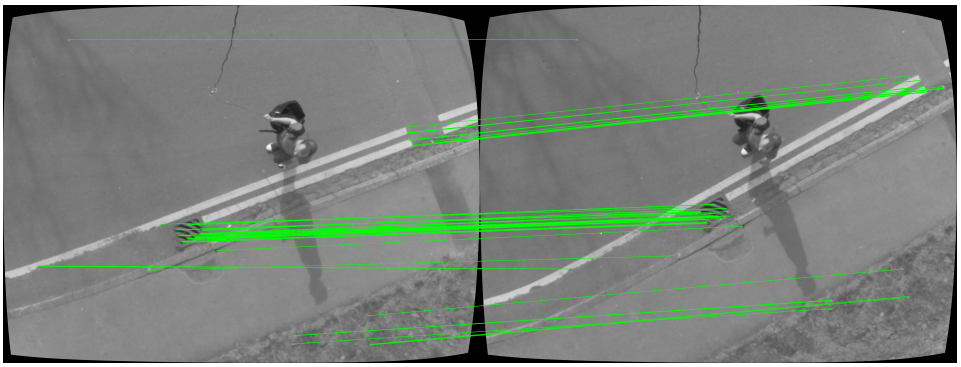


Figure 5.3: Matched points

Despite the best efforts of the brute-force matcher, there are likely to be mismatched points between consecutive frames (especially where there is only a partial overlap). In order to remove erroneous matches the software uses three steps:

1. The relative location of matched points

2. Lowe's ratio test

3. Post-stitch properties

The relative location of matched points is used to remove those matches whose distance between points differ significantly from the mean by a certain number of standard deviations. Because of the nature of the top-down view, the distance between points in images is likely to be of a similar order.

Lowe's ratio test [94] uses the Hamming distance of the two best matches and the hypothesis that any correct matching will be where these two distances will be significantly different (i.e. the best distance will clearly stand out) to the next nearest (presumable incorrect) match, the ratio of these two values are then used to remove erroneous matches. This involves a threshold, matches are removed where the ratio is greater than or equal to 0.8, as with the original work.

The ten best matches in terms of match distance are then used to generate transformation matrices which are subsequently used to verify that the physical size of the output stitched image is not much greater than expected (previous mis-stitches were significantly larger) and that the points in the two images stitch actually overlap. If there are sufficient matches that pass the size and overlap tests then the transformation matrices are stored for future stitching. An example of two consecutive aerial images and their matched points is shown in Figure 5.3.

5.3.3 Stitching and locating

Stitching is initiated by the request from the GroundPi of an update image, upon receiving a request the most recent N (2) images are stitched and transformed such that they are a planar (x, y) transformation from the original input image at the same scale using matrix multiplication and the stored transformation(s) from start to current image. The most recent of the location updates is also transformed using the same matrices to get the current location of the vehicle in real space.

5.4 Texture-based segmentation

Much prior work on traversability based on overhead imagery is particularly vague about what constitutes "risk". Typically this is either risk of colliding with an obstacle or risk of not being able to traverse a particular region.

This work uses segmentation based on the concept that an area is traversable if it is similar in some way to the area in which the vehicle commences its route. Assuming that the user initialises the robot in an area is safe, that area and other similar looking areas are likely to be safe. One limitation of the texture-based similarity approach is that overlapping areas with similar texture but with unsafe and unclear borders may be misclassified as safe (e.g. steps as the aerial imagery won't show depth). Another limitation is that safe areas such as road markings may be misconstrued as unsafe as they don't match the surrounding road area texture. A measure of texture based on the GLCM values of correlation, contrast and entropy is used to show similarity (with appropriate speed improvements to enable real-time usage). Once similarity maps are generated using the combination of correlation, contrast and entropy they are thresholded in order to identify traversable areas.

Colour-based segmentation was trialled using similarity of colour and thresholding (manual and dynamic using OTSU) but discounted due to lack of robustness in changing environments. When looking at texture, various methods are available including local binary patterns [116] (LBPs), Law's texture energy measures [64] and grey level co-occurrence matrices (GLCMs). LBP's have typically been used for face detection and are comparatively fast but are more suited to precise (close-up, short range) applications. Laws texture measures generate texture values dependent on a series of masks. Both LBP's and Laws measures were ultimately discounted because of poor performance in the initial evaluation and lack of flexibility in favour of GLCMs.

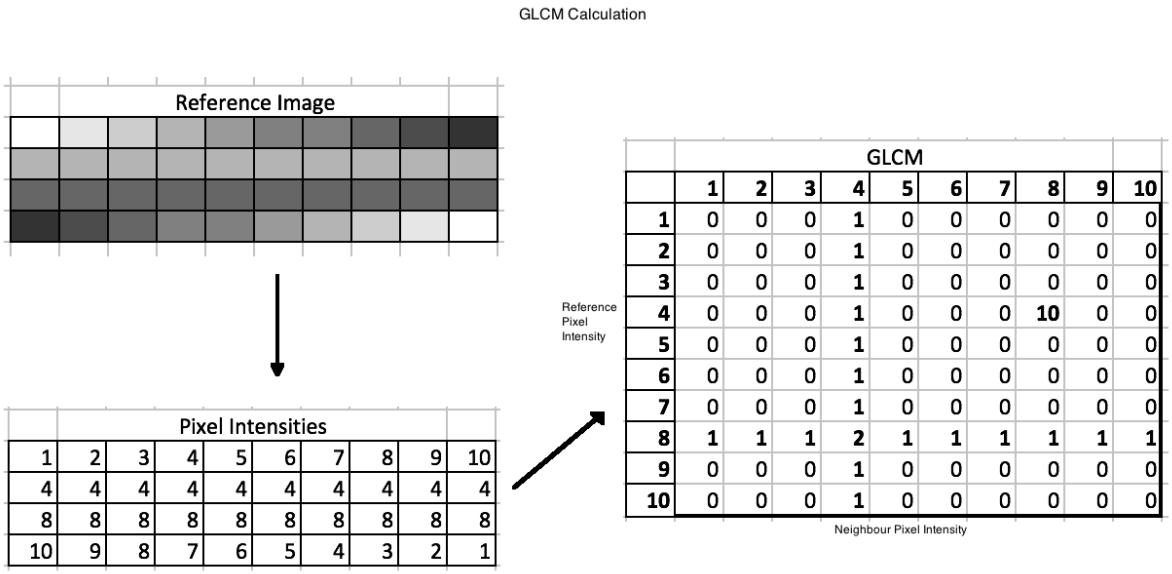


Figure 5.4: GLCM generation example (reference pixel immediately below current pixel)

5.4.0.1 Grey level Co-occurence Matrices

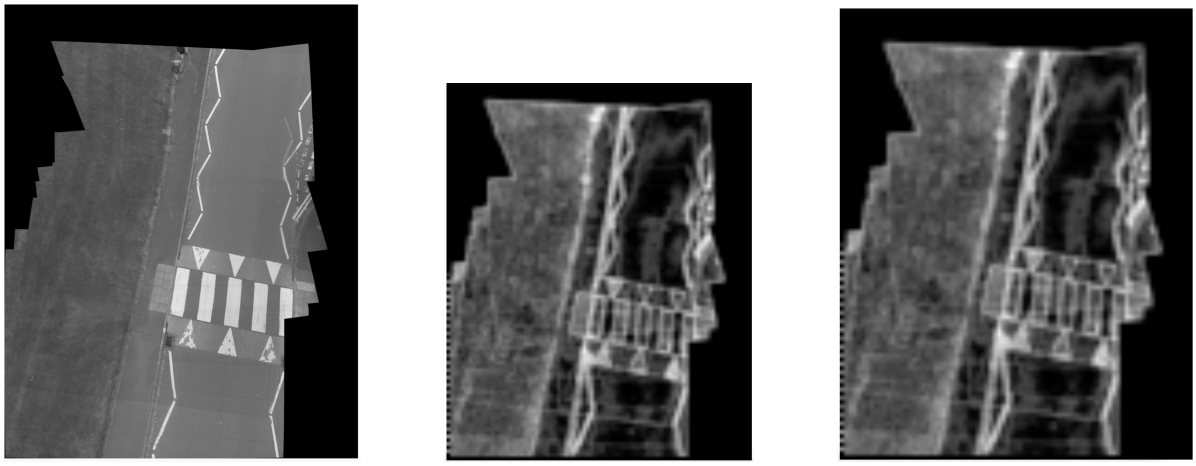
GLCM's generate data using the sum of pixels of a specified relationship over an image (see Figure 5.4 showing an example of GLCM generation with the reference pixel immediately below the current pixel).

This work used the GLCIA[85] (A hybrid method that computed sum and difference histograms using the lookup hash tabled structure but used the GLCHS where sum and difference were unable to compute the required statistics), significantly reducing computational time. In order to generate traversability as a value in real-time, a combination of the contrast, correlation and entropy values as proposed by Clausi[77] (when evaluating different combinations of Haralick's texture features) was used as the basis of the texture measure.

5.4.1 Texture generation

After stitching N frames as detailed in Section 5.3 and generating images (see Figure 5.5a), the image is converted to a texture-based image (Figure 5.5b). Texture-based images are generated using a speeded up GLCM (grey level co-occurrence matrices as proposed by Haralick [62]). The GLCM structure represents texture as frequencies of corresponding pixel values at a certain orientation from another pixel, and a series of features have been proposed to be able to quantify data from the GLCM [76].

In order to speed up execution, as well as limiting the GLCM size (by limiting the number of pixel value bins), the speeded-up GLCHS and GLCIA structure(s) were used, these combine



(a) Stitched image

(b) Texture image

(c) Transmission downsampled texture image

Figure 5.5: Texture generation images

the two-dimensional array structure of the GLCM with a linked list, significantly speeding up data access. After populating the GLCM-based structure, a combination of Correlation, Contrast and Entropy are calculated in order to generate the texture image.

The resultant texture image is generated using a sliding-window that calculates the combined texture value for an area, writes this to the output image and repeats until the whole of the stitched image has been converted to a texture-image. In order to save time, the texture area is measured as a 15×15 pixel zone which on the output image is given the intensity of the combined texture value. A side effect of the use of 15×15 pixel texture areas is the effective downscaling and generation of blocky images formed of squares of groups of pixels representing texture areas. The disadvantage these downscaled images is that resolution is lost, however, significant savings can be made in transmission time by using downsampled images (See Figure 5.5c) and then re-constituting the images at the other end without data loss.

5.5 Threading and synchronization

Because the hardware (Raspberry Pi v3) is a multicore processor we have the ability to reduce time by threading the various processes. In the FlyPi there are two main threads in operation that operate independently (subject to some resource locking):

- Capture, Detection, Movement, Feature Detection
- Stitching, Texturing, Transmission

Table 5.1: Roundel detection - Successful detections in images

Roundels detected	3	2	1	0
Grass background (%)	31/100 (31%)	35/100 (35%)	22/100 (22%)	12/100 (12%)
Tarmac background (%)	23/168 (14.02%)	53/168 (32.31%)	56/168 (34.14%)	32/168 (19.51%)

The capture and transmission thread captures the images, detects the target, updates the motor controller and finds the transformation to the previous images. The transmission thread, after receiving a request from the GroundPi attempts stitching and texturing before sending the data if possible. Synchronization and locking are required in order that data is not altered between capture and transmission.

5.6 Testing and discussion

Testing the FlyPi unit by itself (not the communication and interaction between it and other hardware) consisted of evaluating the performance of the following features:

- Target detection
- Keeping above the target
- Stitching
- Texture and colour-based segmentation

Threading and synchronization are not evaluated separately.

5.6.1 Target detection

The aim of target detection was to detect the target in each frame, these tests were performed on the previous revision (Raspberry Pi 2) hardware.

To evaluate the roundel and target detection system 400 grass-background and 400 tarmac-background images were captured outdoors, with varying illumination and wind speed. The targets were not present in all frames: 100/400 grass-based images contained the target, and 168/400 tarmac background images contained the target. The performance of the system is shown in Table 5.1, which reports the number of roundels detected in the frame in situations where the target was present.

In practice, the detector did detect at least part of the target in approximately 80% of cases. After detection, the grouper groups the potential roundels into targets and selects the best group where there are multiple candidate groups (Table 5.2). Although the algorithm

Table 5.2: Target selection and grouping

Correct group size	Selected correctly	Selected incorrectly
3	36	8
	9	1
2	34	27
	27	2
1	28	29
	20	0
0	531	48
Total	685 (85.625%)	115 (14.375%)

suffers from a relatively large number of false positive roundel detections at the first stage, these often lie within the bounding rectangle of the target and thus have little effect on the overall target location. There are minimal false positive detections where the target is out of frame.

The overwhelming (685/800) number of targets in frame were grouped and selected properly, those which weren't being largely caused by mis-grouping of anomalous detections from the detection step, particularly where they were located in similar areas and at similar orientations to the roundels in the target. The majority of anomalous detections were ignored when grouping.

5.6.2 Keeping above the target

Given the positional information of the Helikite, a P (proportional) controller was initially trialled (Table 5.3) using the physical distance in metres from the centre of the image. These trials took place in minimal wind conditions (about 2.5mph) but typically failed to keep the target in frame over multiple images. The simple P algorithm failed to keep the target in view in the majority of frames because changes in wind speed, direction, or direction of movement of the target caused a loss of the target due to either over or underpowering of the Helikite. A PID-based controller was considered but was discounted due to the irregular time between updates (captured frames with detected targets). The failure of the P controller, and the unsuitability of the PID controller lead to alterations consisting of a motor power increase due to non-detection, reducing the motor value if located ahead of the target and a cut-off after a certain number of non-detections. After a purely proportional system was evaluated, the alterations below were trialled with a P value of 4%, and results with different values are shown in Table 5.4.

1. The P algorithm typically reacted poorly in changing wind conditions, either being over

Table 5.3: Proportional P flight tests

Target movement (Y/N)	P gain	Percentage of frames with target in view	Average sequence length (frames)
N	0	4.8%	2.52
N	0.01 (1%)	5.87%	3.35
N	0.02 (2%)	19.12%	4.78
N	0.03 (3%)	34.25%	4.56
N	0.04 (4%)	20.06%	3.56
N	0.05 (5%)	16.66%	3.03
Y	0	3%	1.8
Y	0.01 (1%)	3.66%	2
Y	0.02 (2%)	4.5%	2.11
Y	0.03 (3%)	10%	2.4
Y	0.04 (4%)	6.66%	1.81
Y	0.05 (5%)	6%	2.4

Table 5.4: Improved flight tests

Frames per second	Motor addition for non-detection	Ahead of target motor negation	Max consecutive non-detections	No of frames	Percentage of frames with target in view	Average sequence length (frames)
1	3%	15%	N/A	200	17.5%	2.44
1	2%	10%	25	200	1%	2
3	1%	5%	50	500	10.4%	7.09

Table 5.5: Final flight following tests

Wind speed (mph)	**Target movement (Y/N)**	**Motor power (Y/N)**	**P gain**	**Number of frames**	**Percentage of frames with target in view**	**Average sequence length (frames)**
0	N	N	0 (0%)	500	24.53%	9.43
0	N	Y	0.04 (4%)	500	4.46%	2.79
~2.5mph	N	N	0 (0%)	500	2.86%	8.6
~2.5mph	N	Y	0.04 (4%)	500	**56.13%**	10.65
0	Y	N	0 (0%)	500	1.26%	1.72
0	Y	Y	0.04 (4%)	500	27%	3.13
~2.5mph	Y	N	0 (0%)	500	10.4%	5.03
~2.5mph	Y	Y	0.04 (4%)	500	11.06%	2.40

(located ahead of the vehicle) or underpowered (located behind the vehicle). A stepped search was implemented that increased the motor power if the proportional motor power component was insufficient to keep the target in the frame (at 3% per non-detection), reducing the motor search value if the target had been overshot (15%). The proportional gain parameter (P) was retained from the previous target detection.

2. Unfortunately, the residual P and search step led in some cases to the Helikite overshooting without a chance to detect the target, resulting in full motor power and a loss of the target. To counteract the loss ahead of the target, a maximum consecutive frames parameter was added to reset the non-detection component after twenty-five successive non-detections. At this point, the P value was reduced to 2% and the overshooting reduction reduced to 10% to compensate for the resetting due to non-detection.

3. Due to the release of updated hardware, and the desire to increase update frequency, the hardware was upgraded to a Raspberry Pi V3. The upgrade had the effect of trebling the frame rate, after which the search variables were altered accordingly.

Increasing the computational speed allowed for more frequent positioning, however, in stronger winds, the search phase was typically too short to find the target. When the search succeeded in finding and moving ahead of the target the sharp motor cut caused by the reduction value tended to cause a subsequent loss of target as the kite fell behind the vehicle again. This resultant oscillatory effect required the control parameters to be altered. The current parameters add 1% to the motor power for each non-detection, reduce the motor power by 5% for each detection ahead of the centre line and search for a maximum of 75 frames consecutively. Results from the most recent tests run both with and without motor power in windy and indoor conditions are shown in Table 5.5.

The results of the motor power testing varied, without any motor power and with movement (either caused by movement of the target vehicle and subsequent towing, or wind), there were minimal frames in view. The minimal number of frames in view during these baseline tests is likely caused by air movement and subsequent lift combining to place the kite at 45 degrees from the ground tether, and because of the relatively small FOV, the ground target being out of view.

Where there was little or no movement of either the air (wind) or the ground-vehicle, there was a significant decrease in the number of frames in view (from 24.53% to 4.46%), this is due to a lack of focussed Helikite direction without wind or movement and the subsequent circling of the target when motor power was applied as opposed to just floating above the target when unpowered.

Where there was wind but no movement, there was a clear increase (from 2.86% to 56.13%) of frames in view, the wind giving appropriate direction to the Helikite and the motor power, keeping it above the target. Likewise, when there was movement but no wind, the air movement caused by forward motion increased detection (1.26% to 27%). Unfortunately, when there was both wind and movement, there was a negligible increase in detection, which was likely more due to luck than the algorithm itself. Anecdotally, the combined wind and physical movement seemed to overpower the efforts of the selected motor and as such the Helikite was always trailing the ground vehicle.

In terms of success, in its raw form, the performance in no wind/movement and too much wind/movement was unsuccessful but the performance in some wind or movement was successful with a clear, repeatable increase in the number of frames with the target in view. In practice, the motor would benefit from being more powerful though this has implications in terms of weight and power consumption. The speed of the ground vehicle is constrained by other factors (such as stitching and mapping), with wind speed having a greater effect on the motion of the Helikite and subsequent keeping of the frame in view.

Tests of the current algorithm have shown the following parameters to be related to performance:

- P Gain: The proportional value applied when the target is in frame, lower values cause the Helikite to become lost behind the target and re-enter the search phase whereas higher values cause the Helikite to become lost ahead of the target.
- Motor addition for non-detection: The amount of extra motor valued applied per frame of non-detection, smaller values cause slower or no target finding, larger values cause erratic movement, faster target detection and possible overflying.
- Ahead of target motor negation: Reduction of the additional motor value when the target

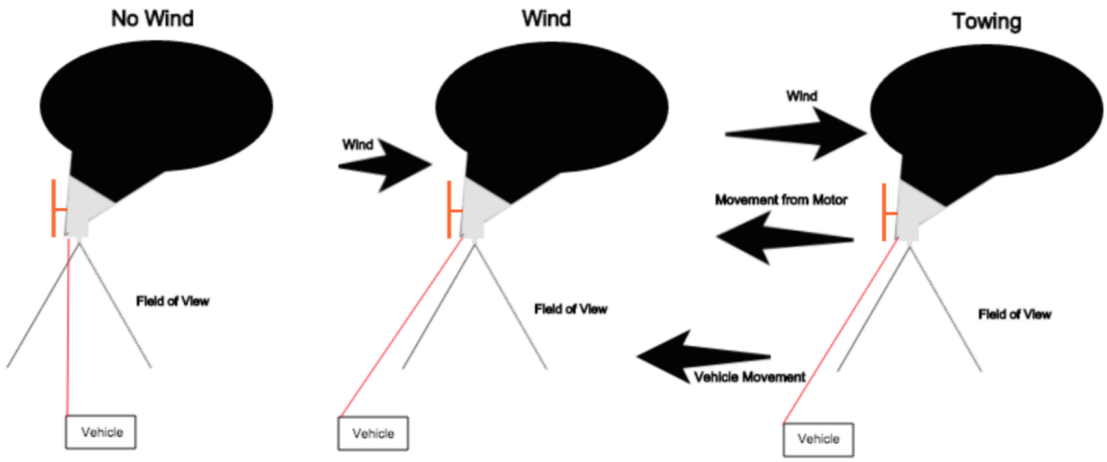


Figure 5.6: Helikite flight characteristics

is passed, smaller values increase the risk of overflying whereas larger values increase the rate of motor cutout and subsequent loss of the target and re-entry of the forward search phase.

- Reset value: The number of non-detected frames in a sequence before the additional motor value is reset, a low value causing the search to fail and high causing the Helikite to be lost ahead of the target.

```
input: A binary image
buffer[5], index = 0; // buffer storing ring widths of a potential target, index
    of first free space in buffer
vector <targets> potentialtargs; // storage of candidate roundels
int stripewidth; // width of current sequence of coloured pixels
for i ← 0 to height do
    for j ← 0 to width do
        int currentcolour = pixelvalue.at(j,i); if pixelvalue.at(j,i);==pixelvalue.at(j-1,i); then
            // If the current pixel value is equal to the previous pixel
               value then sequence continues
            stripewidth++;
        else
            // Else swap current pixel value and the sequence to the end of
               the buffer
            currentcolour= pixelvalue.at(j,i);
            buffer[index]=stripewidth;
            index++;
            if index==5 then
                if currentcolour==255 then
                    // Because the input of the image is binary (either black
                       or white), a value of 255 is acceptable given noise is
                       removed in the binarisation step
                    // If the colour of the last sequence was black and the
                       buffer is full, start of circle is equal to current x
                       location minus the circle width (sum of buffer)
                    int start = j-(SUM of buffer[]);
                    int end = j;
                    for All of the values in buffer do
                        if The sequence width values stored in buffer all have the same ratios
                        to each other as in an ideal target (+/-) error percentages then
                            // Add to the list of potential vertical targets
                               verified horizontally but not yet vertically
                            updateTargets(start, end, buffer[2], y, true, false);
                        else
                            // Add to the list of potential vertical targets
                               not verified horizontally or yet vertically
                            updateTargets(start, end, buffer[2], y, false, false);
                        end
                    end
                end
            end
            shiftdown(buffer);
            // shift the buffer down so that size is equal to five
            index=4;
            // buffer[4] holds most recent (outer) ring width
        end
    end
end
```

Algorithm 1: findhorizontalTargets: Scans horizontally and selects target sequences of pixels

```
input : horizontal start of roundel sequence, horizontal end of roundel sequence, current y, vector
        of Targets (roundels) to check, Bool haspassedhorizontally, Bool haspassedvertically
// Takes inputs from the detection methods including whether or not they
   have passed horizontal and or vertical checking
// Target data structure holds leftx (x location of roundel start), rightx
   (x location of roundel end), centerx, firstmaxy (first y of maximum
   width) and lastmaxy as integers and horizchecked (whether its been
   checked horizontally) and vertchecked boolean values.
for i ← 0 to Targets.size() do
    // For all of the targets
    if centre of current roundel is close to Targets[i] centre AND haspassedhorizontally ==
    Targets[i].horizchecked then
        // Expand existing target(roundel)
        if Target[i] Diameter < end-start(detected target width) then
            // Expand existing target(roundel) horizontally
            Target[i] leftx = start;
            Target[i] rightx = end;
        end
        if Target[i] Diameter == end-start(detected target(roundel) width) then
            if y<firstmaxy then
                // comes before first max y in group
                firstmaxy=y;
            else if y>=firstmaxy AND y<=lastmaxy then
                // comes between first max y and last max y in group
                do nothing;
            else
                // comes after first max y in group
                lastmaxy = y;
            end
        end
        centerx = rightx+(rightx+leftx)/2;
    else
        // Add new target(roundel)
        Targets.add(Target(y, start, end));
    end
end
```

Algorithm 2: updateTargets: Adds a candidate roundel sequence to the stored lists of candidate roundel sequences

Chapter 6

GroundPi

6.1 Introduction

The GroundPi unit is a Raspberry Pi based mapper, planner and movement controller. It takes a series of texture-based images from the flying robot vehicle (FlyPi) and uses them to generate a similarity-based map of the area around the ground robot. Using the generated map a path between the start and goal points is planned using an RRT-based planner and executed in a closed-loop fashion. Traversability and locational information are gathered from the update images transmitted by the FlyPi. This section details the design choices whilst developing the GroundPi controller.

6.2 Path planning towards a goal

Given appropriate inputs of texture images and locational information from the FlyPi system, the GroundPi unit plans and if necessary replans the path from its current location to the goal state.

6.2.1 Selection of a goal state

The original goal of the project was to design and implement a system to control a robotic waterborne vessel travelling within a pre-specified range of the face of a glacier constraining the plannable area to a band between the closest and furthest distance allowed. With the change to ground rather than water-based platforms, there was a shift towards selecting a new goal state. Two different types of goal state were considered:

- Point to point planning: Planning between two points (start and goal) with the aim being

to reach the goal point. This has the advantage of supporting a scanning survey path, through the construction of a sequence of goal points within the area to be surveyed.

- Feature-based planning: Planning from a start point and then either passing through areas with a desirable metric, avoiding other areas based on an undesirable metric or a combination of both. This variety of planning is less structured and more exploratory.

Both options were initially considered, however, point to point planning was ultimately chosen because of its simplicity. Feature-based planning whilst more interesting would have required the selection of multiple metrics e.g. desirable/undesirable areas, their relationships to each other and how they are evaluated to select a suitable path. Additionally, feature-based planning does not easily support survey paths.

6.2.2 Inputs to the planner

The input to the GroundPi system consists of an initial relative position representing the goal from the start location and incremental inputs of both images and location from the FlyPi.

The goal input at runtime consists of a relative x and y distance from the goal location to the start location in the forward direction of the robotic vehicle. The input of the goal state is in relative distance as opposed to global coordinates because of the relatively small distance covered, possible error introduced by lack of resolution of global coordinate representation and the requirement of global-coordinate based planners to know their real-world location (and consequent hardware requirements such as RTK-GPS). A global coordinate based system was implemented but not used due to the relatively small distances and indoor nature of many of the tests.

The FlyPi provides updates at irregular intervals. The time between updates depends on the speed of detection and matching as well as the time taken to transmit which can vary depending on factors such as interference and data size. Considerations when selecting a map-representation were processing speed and ease of planning. Topological (graph-based) map representations were discounted because of the comparative complexity of generation despite their efficiency in planning and storage.

The map representation chosen was a two-dimensional grid-based traversability map similar to an occupancy grid in real-space with a cell resolution of 1cm. Each grid cell contains a value of similarity (see Figure 6.1) between 0 and 1 (1 being completely traversable, 0 being untraversable). Uncertain values are represented by 0.5 where an area of map has not been initialized/updated.

Map size is based on the distance between the start and goal locations such that both are on the map with an additional buffer of two metres. Cells are initialised to 0.5 (uncertain).

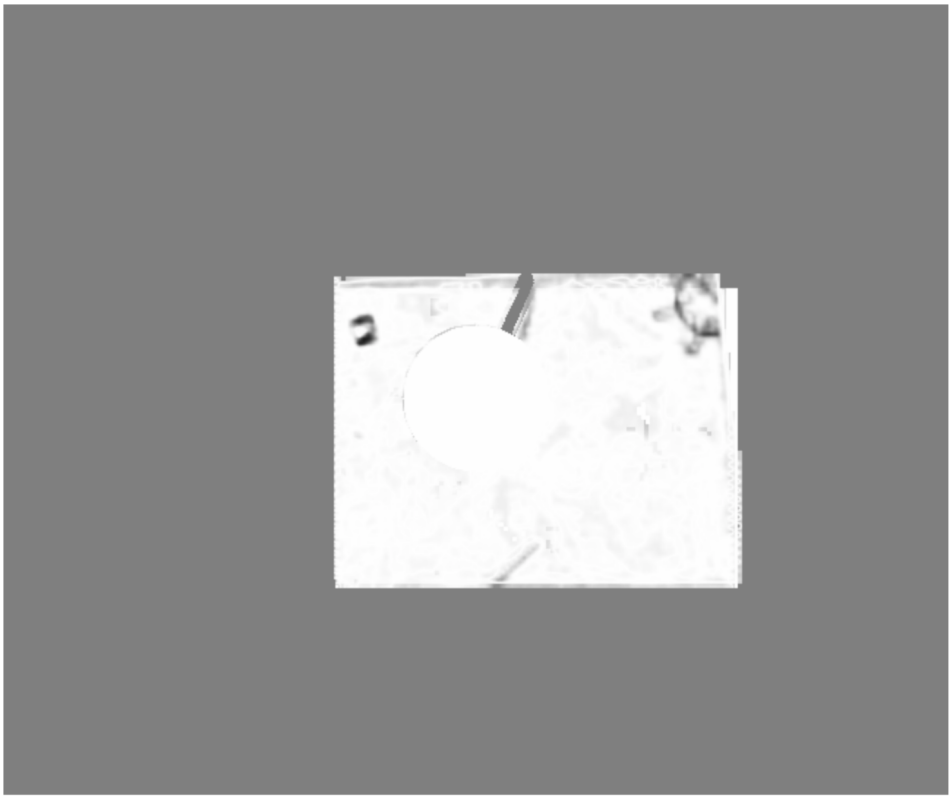


Figure 6.1: GroundPi map example. Grey = unmapped, white = traversable, black = non traversable

A similarity value (Section 6.3.2) is used to update the map using a Bayesian update step to remove noise[117].

6.2.3 Path planning requirements and output

Path-planners have various attributes, the key requirements for this application are:

- Speed: How quickly a path can be planned.
- Optimality: How optimal the resultant path is or is guaranteed to be.
- Completeness: Whether or not the planner is guaranteed to find a path if one exists.
- Re-Planning: Whether the planner is capable of partially re-planning part of the path given updated information.

The strongest consideration was speed, followed by the ability to re-plan given the real-time requirement and likelihood of incomplete data. The ability to partially re-plan could be ignored if a planner was so fast that re-planning the whole path was possible in real time making partial re-planning unnecessary. Completeness or an approximation of it was desirable although optimality was deemed unlikely at the outset given the lack of a fully updated map. A "good enough" solution is therefore acceptable. The planner outputs a list of safe points to be traversed by the robot vehicle.

6.2.4 Planner selection

Consideration was given to identifying and implementing some kind of "braveness" based planner that could theoretically alter its level of acceptable risk based upon some metric. Unfortunately, in the real world terms such as "braveness" or "risk" are fluid and often subjective. Much prior work tends to simplify and replace risk with a single likelihood of a negative event such as collision or non-traversability, or with the inverse positive likelihood of non-collision or traversability, although some other measures are proposed in Section 6.3.1. Some methods explicitly define an overall "risk" such as work by Shan and Englot [118] that used an RRT with two user-defined cost criteria and a cost hierarchy to prioritise risk avoidance although this is complicated and requires extensive tuning to define risk types and priorities. Given the work required to select, implement and tune a planner, it was decided not to use a fluid level of risk because of the increased complexity, Instead, it was decided to simply implement a binary planner with a pre-selected risk threshold based on the concept of traversability.

The sampling-based RRT was chosen because although not optimal, it is probabilistically complete and much more applicable to real-time usage than either the A*, D* or D* Lite

algorithm. A*, D* and more recently D* lite algorithms are typically used where the entire configuration space is known before planning and the situation is therefore less likely to require time-consuming re-plans. The data from the FlyPi unit, however, is unlikely to be complete and so re-plans are likely to be necessary. Knowing this, a sampling-based planner was chosen because of its speed. RRTs were picked over PRM because of their ability to handle non-holonomic or kynodynamic constraints.

The original RRT [17, 18] has been extensively evaluated and modified since its inception to improve speed, optimality and completeness. Most of the prior alterations were redundant in this project given the comparatively small search place, probabilistic completeness and the "good enough" paths generated by the original algorithm with basic alterations. Pruning the original RRT gave a path that is acceptably close to the optimal path where optimality is measured in terms of distance travelled.

6.2.5 Alterations to the RRT algorithm

Some alterations were made to the original RRT algorithm [17, 18] to allow real-world planning. The changes made to the original RRT seek to fulfil the requirements in Section 6.2.3 and consist of the following:

- Robot Parameters: When planning, the size and shape of the robotic vehicle are considered when evaluating safe areas i.e.there should be sufficient safe space to accommodate the vehicle.

- Pruning/smoothing: Once an RRT-based path is generated, the path is pruned in order to shorten it, this is achieved by repeatedly checking direct free paths between non-consecutive sub-points and removing intermediate points in order to straighten convoluted paths. The number of each pruning/smoothing cycles is currently set at 1000 which gave acceptable performance in terms of time and path reduction.

- Bi-Directionality: In order to improve the speed and efficiency of the RRT algorithm, LaValle and Kuffner proposed a modification [21] to the original algorithm that instead of growing a single tree from the source to the goal or vice versa, grows a tree from both towards each other. The bi-directional tree-growing has been shown to converge more quickly to a solution and also helps to prevent failure when tree-growing becomes trapped in local minima.

- Goal Bias: The RRT accepts a value (goal bias) that will choose the goal instead of a random point in space to grow the tree towards (thus allowing free space exploration whilst growing towards the goal). Setting the goal-bias value too low typically causes

a larger tree than is necessary (as the planner is not planning towards the goal in most cases but extending the tree into free-space) whereas setting the value too high will typically cause the planner to take longer (as the tree is not extended enough into free-space). The value has been set at 10% i.e. statistically 1 in every 10 times the planner will attempt to connect the goal to the nearest point in the already-grown tree as opposed to connecting a random point in the free-space to its nearest neighbour point in the tree.

6.2.6 Evaluation

The original RRT[19] algorithm has been proven to be probabilistically complete and fast. Section 6.3.3 supports the previous literature in that the RRT-based planner will produce an acceptable "good enough" path when using an appropriate goal bias value that is also approximately distance-optimal in best cases. In terms of speed, the RRT algorithm appears to converge to a solution in a negligible amount of time and therefore negate the re-planning requirement, though in practice partial replanning is used to minimise path disruption.

6.3 Path evaluation

Given textural input images from the FlyPi, the map is generated using the idea that the start location is likely to be safe and therefore similar locations are also likely to be safe. When generating the initial map, the area around the target vehicle is evaluated and a value for safe texture is obtained, this is used in conjunction with a minimum similarity threshold (T) in order to generate planned paths.

6.3.1 What are safety and risk?

In order to use risk as a metric, it first needs to be defined in such a way that it is usable as a means for navigation. Humans primarily judge risk in their everyday lives based on their feelings (instinctive and intuitive reactions to danger) quickly and automatically as opposed to analysing risk based on logic, reason and scientific deliberation. Intuitive reactions to danger will be based on past experiences over the course of the subjects life to date but are unlikely to be quantified, instead being handled quickly and automatically [119] using subjective measures such as "really risky" or "not risky".

In contrast to the human model, robotic vehicles are likely to have limited knowledge about themselves, limited sensor inputs and limited or no prior knowledge about their interaction with the world. The lack of prior knowledge about their interaction with the world potentially allowing robotic vehicles a more mathematical probabilistic model of risk using

concrete values of risk for action than the subjective labels used by humans in the risk as feelings model.

Perhaps the most common measure of risk is the risk of collision (in both single and multiple-risk systems) in both ground-based [120] and water-based examples [121]. Alternative risk measures proposed include traversability [122], risk of detection [123] and also self-evaluation of optimality [122, 124] consisting of singular or multiple inputs.

This project uses a measure of traversability (how traversable an area represented by a grid-cell is) akin to [122]. This measure of traversability differs in that it uses simply the similarity of a texture measure whereas prior work [122] generates probabilistic cost maps using a combined value of pixel colour and longitudinal slip using trained Gaussian Processes. This work has the advantage that it doesn't require training as it uses a measure of traversability generated from a single input, and uses live aerial imagery in order to simultaneously map and locate the ground robotic vehicle without using feedback from the ground vehicle.

6.3.2 Similarity-based mapping

In order to generate a similarity-based texture map, the concept that the path was likely to start in a safe area was used with the assumption that similar areas were also likely to be safe to traverse. In order to generate the similarity map, a safety value was first obtained, this was subsequently used to calculate similarity during runtime.

The initial safety value and map were obtained as follows:

1. An initial update image is obtained from the FlyPi consisting of two stitched frames and the location of the ground vehicle in frame from the detector.

2. Speeded-up grey level co-occurrence matrices and a sliding window are used to convert the image to a texture-based image using a normalised combination of contrast, correlation and entropy.

3. The normalised image and locational data are transmitted to the GroundPi

4. The GroundPi uses the information about the desired location and map resolution to generate an empty map of appropriate size initialised with an uncertainty value for each cell.

5. The average texture value around the ground vehicle in the initial frame is measured. Given that the radius of the robot is known, the area between this and radius $\times$ 1.5 the radius is used (currently set as 30 and 45cm from the centre of the robot) providing a 15cm wide zone as the basis of the "safe" texture value with the assumption that as

humans we would normally wish to have a certain amount of free space around us (half of the radius) at any time.

6. The average texture value or "safe value" is used add the initial update frame to the image based on each cells difference from the safe reference value obtained in the previous step using a Bayesian update step.

Once the initial map has been populated, the map is continuously updated:

1. A new frame is captured, at each capture if possible it is stitched to the previous frame. The homography between the two frames and previous frames are used to calculate its perspective relative to the original reference frames.

2. The stitched frame and vehicle location are warped relative to the original reference frames.

3. The image is converted to a texture image using the GLCM and a combination of contrast, correlation and entropy.

4. The image is transmitted to the GroundPi.

5. The map is updated given the known position relative to the update frame using the difference from safe value and a Bayesian update step.

6.3.3 Threshold selection

Given the generation of a similarity-based map and in order to generate a "safe" plan using a binary planner, a Threshold (T) is needed that differentiates safe from unsafe areas.

6.3.3.1 Test format

Six images (Figure 6.3) generated by stitching images from previous tests and exhibiting a variety of different ground conditions were selected. The six images were converted to texture values and similarity using various T (traversability) thresholds used for path planning (Figure 6.2). Texture values on the map itself are between 1 (completely traversable) and 0 (completely untraversable), for the threshold selection this was scaled to 255 to 0 representing grayscale pixel values. Grayscale values evaluated were 100-255 in increments of 15, T values below 100 in previous tests typically yielded complete traversability despite obstacles. For each image, five traversable and five untraversable paths were planned, each path being repeated five times. Paths were selected based on the human perception of traversability e.g.

a path between two points on a single road should be traversable, in order to evaluate both the suitability of the similarity assumption and to select a threshold T. Only ideal straight line (A to B) paths were evaluated in order to reduce complexity, and because the purpose was simply to set a traversability threshold rather than plan complicated routes.

6.3.3.2 Aim

The aim was to test the performance of different traversability thresholds using the following metrics in combination with the RRT-based planner:

- Completeness: Will a path will be found if a path exists in the real world?
- Correctness: Will the path be safely planned? Given that all paths should be achievable along a single "safe" surface, do they avoid traversing different materials or crossing unsafe thresholds between materials unnecessarily.
- Optimality: How optimal the planned path is compared to an optimal path planned using a proven optimal planner such as A*. Optimality is measured in terms of path length and number of waypoints.

Given the two above metrics, the following hypotheses were proposed:

1. Given an excessively low threshold, the path will be optimal but incomplete (i.e. a short path will be found even if it doesn't exist).
2. Given an excessively high threshold, a path will either not be found, or will be excessively long (in terms of distance).

6.3.3.3 Results discussion

Acceptable T values should allow binary differentiation between safe and non-safe surfaces. Data from safe and unsafe plans (Table 6.1) shows clear success at a threshold value of 175 both in terms of successful paths planned where a path exists and failure to plan where no safe path exists. Where the similarity threshold is set too low, the traversability map tends to lack distinct image features instead being almost entirely traversable supporting hypothesis 1. Setting the similarity value as high produces a virtually non-traversable map supporting hypothesis 2, despite paths existing. Figure 6.4 shows the resultant traversability images generated at low (Figure 6.4b) and high (Figure 6.4c) similarity thresholds and lack of image features of a source image (Figure 6.4a).

In terms of path distance optimality, there was very little difference over varying thresholds with most successful paths being near to optimal. The maximum path distance over all

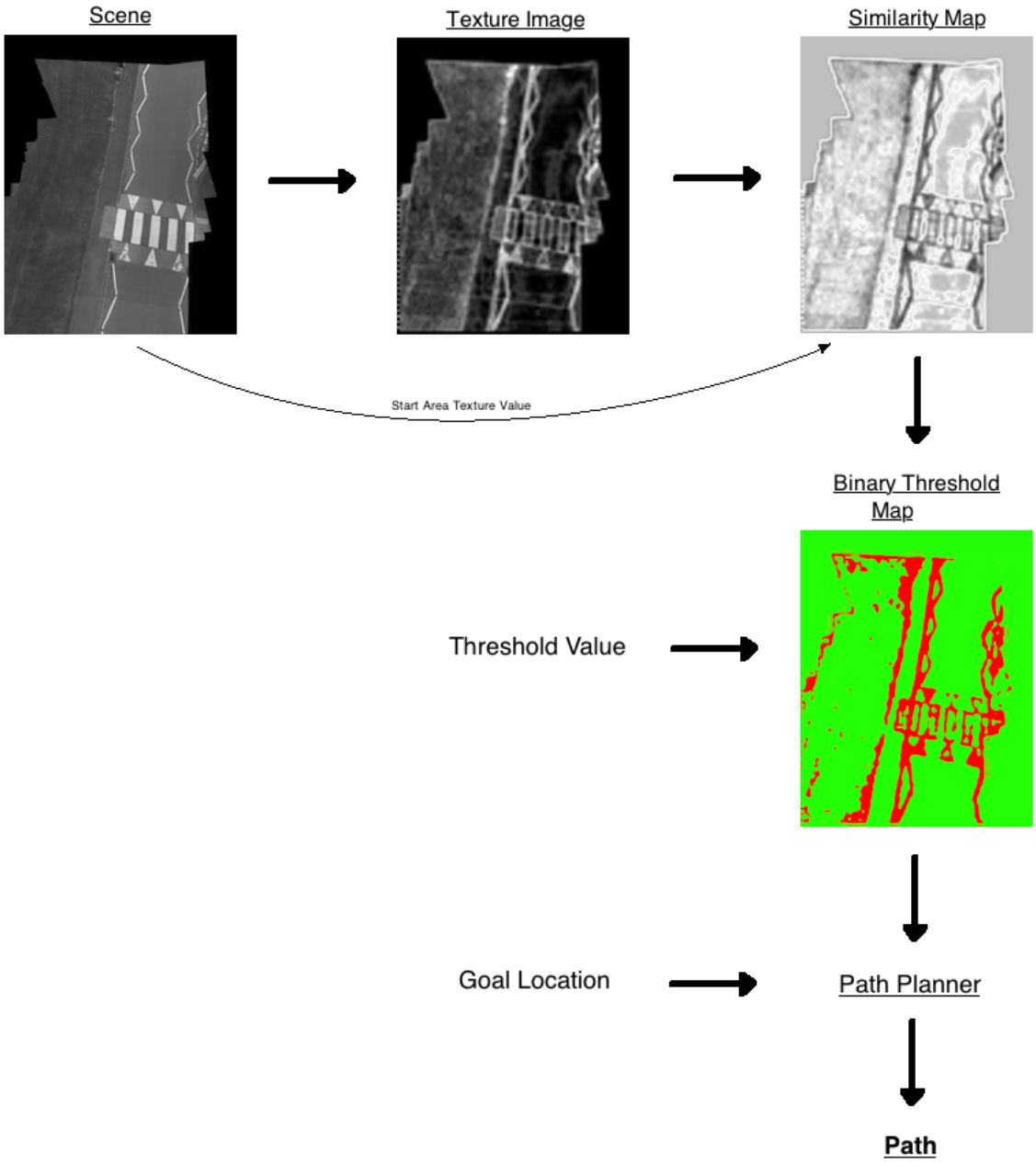
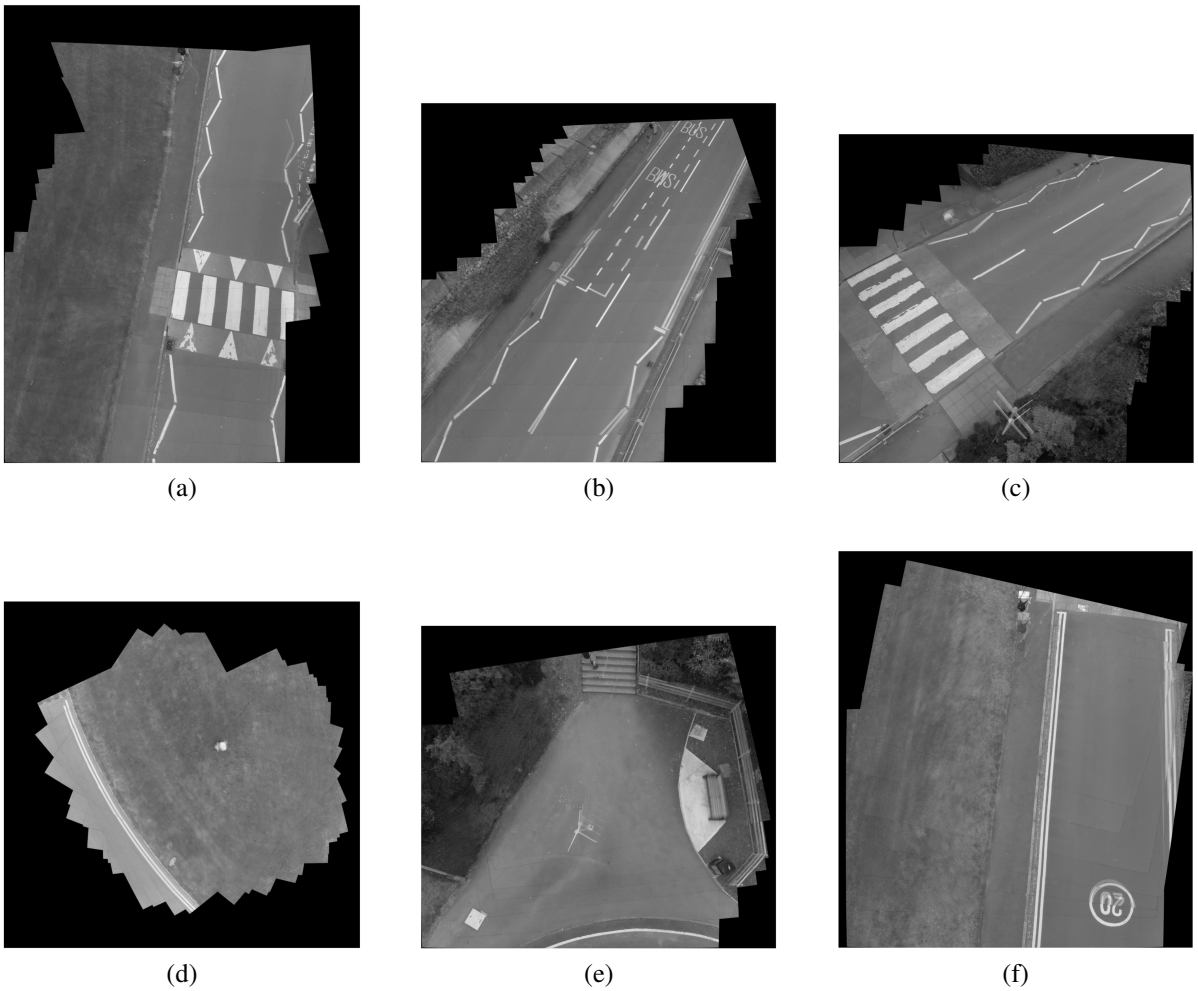


Figure 6.2: Threshold path generation pipeline



(a) (b) (c)

(d) (e) (f)

Figure 6.3: Traversability threshold test images

(a) Traversable source image

(b) Low similarity threshold (T = 130)

(c) High similarity threshold (T = 235)

Figure 6.4: Example thresholded path images

thresholds was 1.116 times the optimal (Euclidean) path distance with a maximum standard deviation of 0.14. For the complete threshold value of 175, the mean distance travelled was 1.027 with a standard deviation of 0.12. In terms of points in the path, the threshold value of 175 had a mean 2.2 points and a standard deviation of 0.68 which is near to optimal given the 2 minimum points (start and goal).

The similarity traversability threshold was set to a rounded and scaled value of 0.7 (175/255 = 0.686) given the success in terms of completeness as well as near optimal performance in terms of distance travelled and the number of points in the path.

6.4 Path execution

Given appropriate path input of a variable-length array of points relative to a local map, the path is executed using a 4-wheel non-holonomic Pioneer 3-AT research robot using skid-steering and interfaced with the GroundPi using the Player library. The movement consists of rotation and forward movement in straight lines. Movements are made between planned points with irregular map and location updates from the FlyPi, the path being re-planned if an obstacle is detected in the path during runtime.

6.4.1 Initial planning

After receiving a goal, the GroundPi starts a map and begins updating using the texture difference values generated in Section 6.3.2, after 5 image updates have been obtained an initial

Table 6.1: Thresholded path planning tests

Threshold (T) Value	Percentage % Of Correctly Planned Paths						
	Image 1	Image 2	Image 3	Image 4	Image 5	Image 6	Mean
100	54	50	50	50	50	50	50.667
115	62	56	50	50	60	50	54.667
130	74	60	70	50	60	54	61.333
145	80	60	100	50	90	78	76.333
160	92	76	100	50	90	96	84
175	100	100	100	100	100	100	100
190	100	100	100	92	90	100	97
205	80	90	70	90	60	100	81.667
220	50	50	50	60	50	80	56.667
235	50	50	50	50	50	50	50
250	50	50	50	50	50	50	50

route is planned. The choice was made to require a certain number of updates to give the initial map the chance to be at least partially populated before planning. A time limit is also placed on the updating step such that the process is terminated if updates are not obtained quickly enough. Termination based on a time-limit would more than likely be because the vehicle is out of view of the camera platform.

6.4.2 Choice of motion planning

The RRT-based planner (Section 6.2.4) simply provides a list of waypoints from and including the start location to the goal location. The Pioneer 3-AT robot is equipped with skid-steering allowing rotation of either or both sides of wheels in order to move right, left or forward. The skid-steering although non-holonomic is assumed to be approximate to it in that the vehicle can rotate on the spot before travelling forwards or backwards in a straight line.

The robot motion is therefore based on turning towards the next target location and then moving in a straight line towards it until the sub-target is reached (see Figure 6.5).

6.4.3 Movement

The movement consists of two-stages (turn and move forwards) between sub-points. At each update from the FlyPi, the location and heading are verified and the path checked for any obstruction which results in one of the following four options:

- Turning required: The robot is not currently facing the next sub-goal, the robot turns

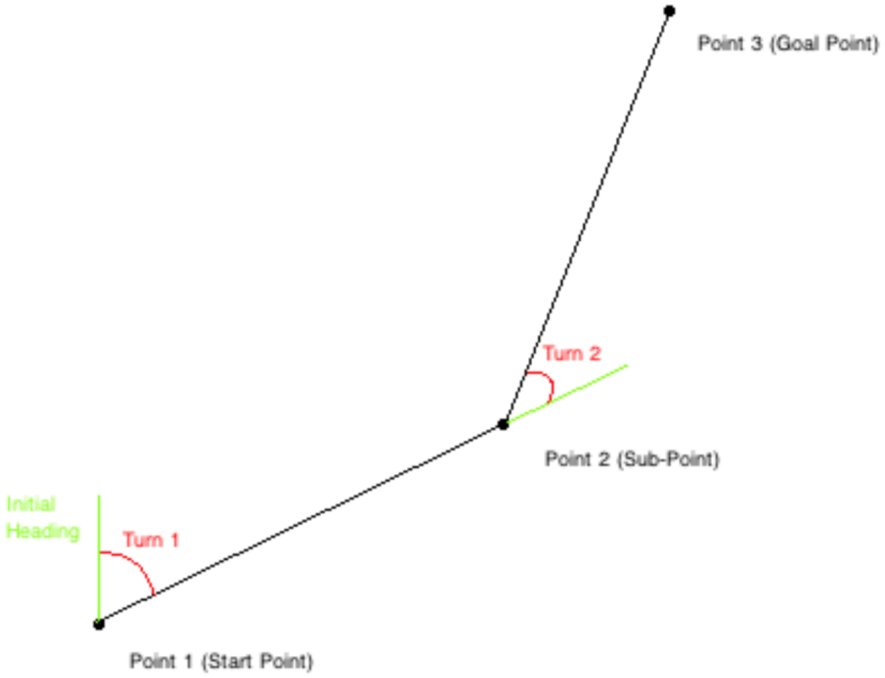


Figure 6.5: Robot motion example

towards the next sub-goal in the shortest direction.

- Moving forward: The robot is facing the sub-goal without obstruction but has not yet reached it, the mover continues moving forward.
- Reached sub-goal: The robot is within an acceptable range of the sub-goal, so the planner resets to move to the next sub-goal. If the sub-goal is the goal, the goal has been reached and movement is terminated.
- Path blocked: The robot is facing the sub-goal but the path is blocked, the path is replanned either partially (if only this sub-path is blocked) or fully (if more than just the current sub-path is blocked).

Because of the time between updates and the low resolution of the robot mover, an acceptable radius of 10cm was set such that if the robot is facing this radius of the sub-goal or has reached this radius then it is facing or has reached the sub-goal as shown in Figure 6.6. On a number of occasions, the vehicle was lost from view when times between updates were extended because of an inability to find a match caused by non-overlap of frames. A further limitation was therefore placed such that if no update is received for a certain time period (5 seconds) then the movement is paused until a match is found. The entire algorithm is shown in Algorithm 3.

```
input: A sorted list of points (the path)
Point path[number of points in planned path]; Point currentlocation, nextpathpoint;
while Not reached goal location do
    // Rotation
    while Not Currently Facing Next Goal-Point do
        Rotate towards next goal;
    end
    if Have reached next goal point then
        // Unlikely because rotating on the spot
        if Current Goal Is Last In List then
            Success - Have Reached Final Goal;
        else
            // Reached Sub-Goal
            New Goal Is Next Goal In Path;
            Move back into rotation loop and rotate towards new goal;
        end
    else if Haven't reached sub-goal but are facing it and path to it is clear then
        // Forward Movement
        while Not reached next goal location, still facing target and path to next goal not blocked
        do
            Move forwards;
        end
        if No longer facing target then
            Move back into rotation loop;
        else if Reached sub-goal then
            if Current Goal Is Last In List then
                Success - Have Reached Final Goal;
            else
                // Reached Sub-Goal
                New Goal Is Next Goal In Path;
                Move back into rotation loop and rotate towards new goal;
            end
        else
            // path to next goal blocked
            Re-plan path, re-start loop;
        end
    else
        // Haven't reached sub-goal but path to it blocked
        Re-plan path, re-start loop;
    end
end
```

Algorithm 3: Mover algorithm

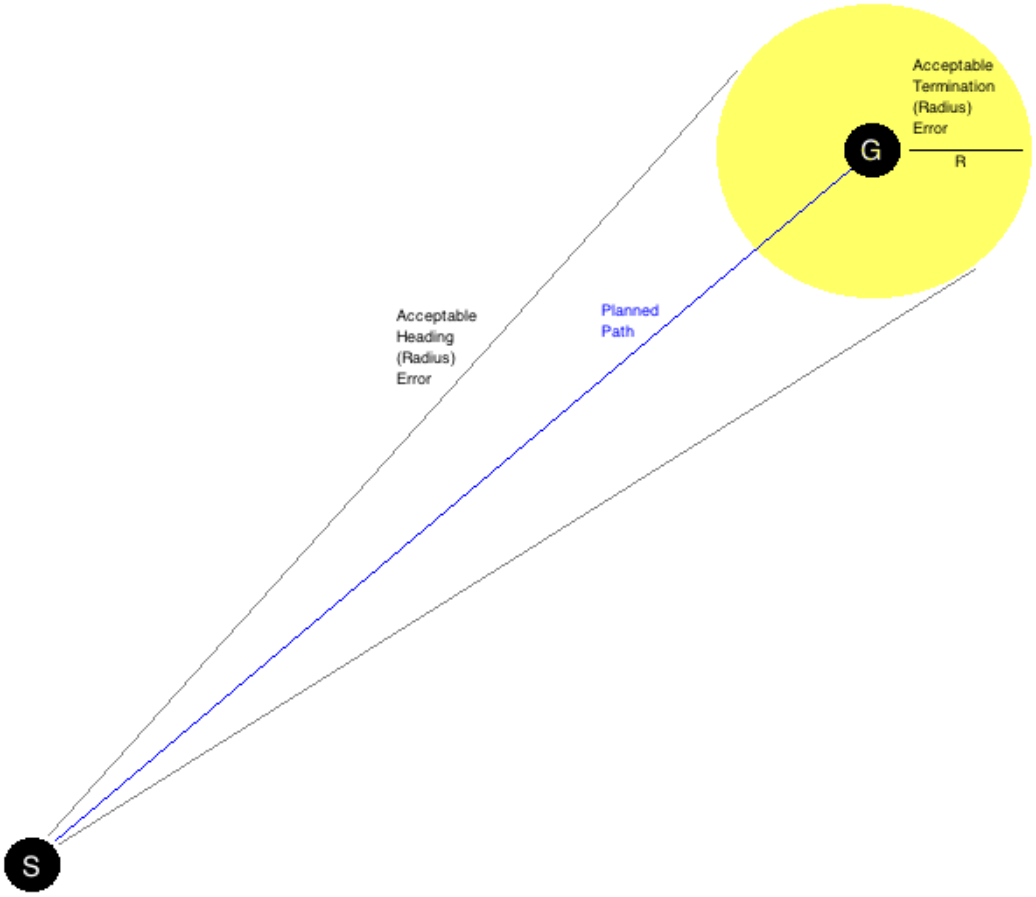


Figure 6.6: Acceptable radius error

6.4.4 Evaluation

In terms of path execution, the selection of the Pioneer 3-AT performs the necessary functions, can interface with the GroundPi unit and is robust and durable (although battery longevity is an issue). The initial plan generated by 5 image updates and the RRT-based planner is in most cases complete and reasonably optimal. Although curved motion plans are likely to be faster in terms of execution time, the distance travelled will be larger and is very much more complicated to calculate with a likely increase in planning time and inaccuracy (due to incorrect motion plans based on unreliable robot movement). The movement does seem to function correctly with the goal being reached in most cases, the time delay between updates, however, causing over-turning and over-shooting on some trials.

Chapter 7

Closing The Loop

7.1 Introduction

Following design, production and testing of the FlyPi and GroundPi, linking of the two systems was completed, the aim being to test how they work together and how the system functions in general. Three series of indoor tests were carried out in order to test not only the ability of the robot to plan and move using an RRT-based path but also to evaluate the practicalities of using start area similarity as a measure of danger. In order to test the coordination between the FlyPi and GroundPi units in an indoor environment, a series of three tests were proposed with varying levels of obstruction type in order to evaluate both the system cooperation and its reaction to obstacles during execution. In addition to the obstacle avoidance runs, a small number of additional tests were made within a very constrained path to confirm obstacle avoidance. After the successful conclusion of the indoor tests, some outdoor testing was performed with the addition of the powered movement system to evaluate the system as a whole in a real-world environment. This section details the results of these tests with the evaluation of performance in terms of path optimality, the time taken and distance travelled.

7.2 Pre-tests

As part of the validation of the system, a series of pre-tests were undertaken with aerial images captured from a fixed viewpoint using the same camera mounted to the ceiling of the test building, rather than the Helikite. The resultant tests evaluated the performance of the transmission, planning and mapping system with a fixed viewpoint. Because of the fixed viewpoint, images were not stitched and therefore the time between updates was shorter with a resultant increase in accuracy due to decreased lag. Although most of the tests completed, there were

instances where the vehicle overshot the desired path and left the fixed visible area, leading to a complete loss of the vehicle caused by latency between image capture and updating of real-world location. Subsequent modifications limited the plannable area when using a static viewpoint in order to avoid loss.

7.3 Indoor test structure

In order to evaluate the performance of the combined system, three series of test were proposed, each with a different type of obstruction:

- No Obstruction: No obstruction between the start and goal points, ideally the vehicle should turn towards the target and move forward until reaching the goal.
- Fixed Obstruction: Obstruction between the initial location and the goal location from the start, the vehicle should navigate around the target and reach the goal.
- Introduced Obstruction: Obstruction is introduced after initial planning and so the system should re-plan during execution.

These tests attempted to evaluate the proposed system in various possible scenarios. The obstruction-free test simply sought to evaluate the ability to plan an initial straight-line route between two points and to execute movement. The purpose of the fixed obstruction test was to confirm that obstructions would be noticed by the texture-based mapping system and that appropriate paths could be planned in real time, taking the parameters of the ground-vehicle into consideration. The purpose of the final set of tests was to confirm that unexpected changes during runtime could be taken into account and that the system could dynamically replan during path execution.

Obviously, the series of tests do not cover every eventually, nor do they seek to, other scenarios could have included moving obstacles, obstacles of different sizes or variations in the shape of the obstacles. Unfortunately, moving or different changes of obstacle sizes were not evaluated due to the complexity of introducing another moving system during runtime, the additional time needed to complete additional experiments, and the increase in complexity of data recording and analysis required when using a moving obstruction. In any case, the three series of tests proposed should prove sufficient to not only measure the coordination between systems and path planner (all tests), the detection of obstacles (fixed and introduced obstacle tests), and the ability to replan during execution (introduced obstacle tests), which would be the features necessary to potentially navigate different sized, different location, and moving obstacles.

Figure 7.1: Test obstruction

Each series of tests (No Obstruction, Fixed Obstruction, Introduced Obstruction) was run in four directions from the start point (45, 135, 225 (-135), 315 (-45) degrees) each repeated five times for a total of 20 runs per test type. For those "runs" that contain an obstruction, the obstruction consists of a "dazzle" patterned board of approximately 15x20cm placed directly between the start and goal points to simulate a vastly different texture (see Figure 7.1).

A variety of data was recorded during the tests, in addition to imagery of captured frames, stitched images and the state of the map during execution:

- Start and goal locations
- Whether the goal was reached
- Whether the vehicle was lost or hit an obstruction
- Terminating distance from goal
- Final location and final distance to the goal
- Time taken
- Number of times that the system replanned during execution.
- Forward movement distance: how far the vehicle moved forward in a straight line during execution.
- Rotational distance: the sum of the rotations of the vehicle during execution
- Relative final location of the vehicle from the start position.

Tests were undertaken in an indoor environment with a single colour grey floor, as a result, the proportional Helikite movement system was not used in order to avoid hitting low hanging structures. Terminating location and distance from goal as well as Final Location and distance from goal are shown in Figure 7.2, with any disparity between the two due to latency in the

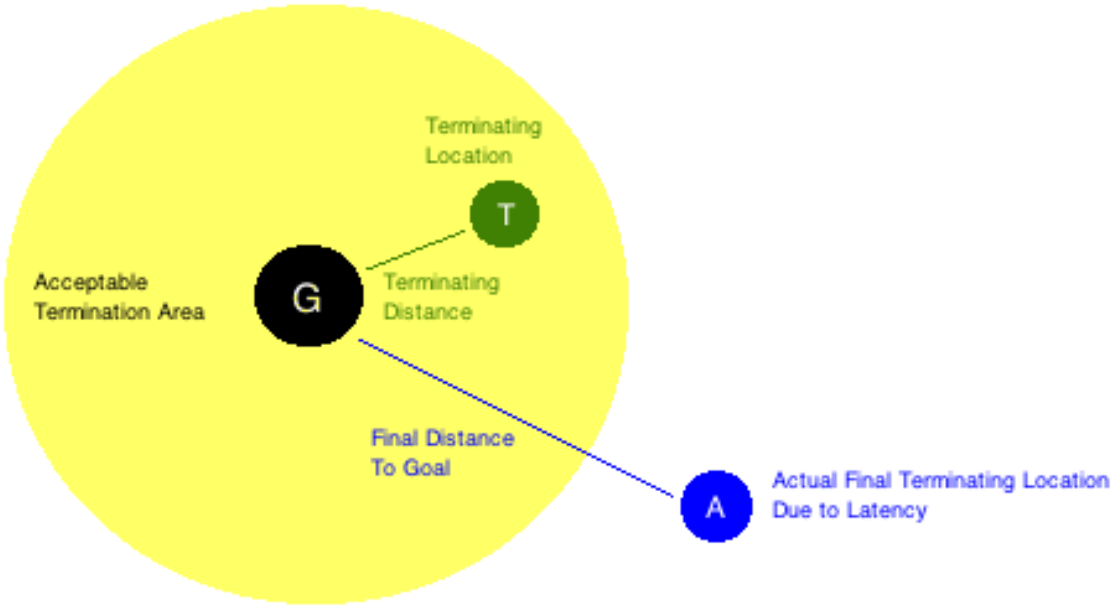


Figure 7.2: Goal locations

control system. An example of forward movement distance and rotational distance is shown in Figure 7.3.

7.4 Indoor data analysis

This section undertakes some data analysis using information gathered from the indoor tests. Suggestions are made to explain the results where appropriate with a particular emphasis on system configuration and changes in experimental setup or parameters and their effect on the results.

7.4.1 Raw performance measures

A series of measures based on the raw data from the test runs.

Start and goal locations: Automatically generated based on the pose of the FlyPi and Helikite with respect to the ground robot and relative goal location. Four goal locations were used at (100,100), (100,-100), (-100,100) and (-100,-100)cm from the ground vehicle in a forward direction.

Goal reached: Whether or not the goal location was reached. In all cases with varying obstruction type or location, the goal was reached. Exceptions to this were when tests were

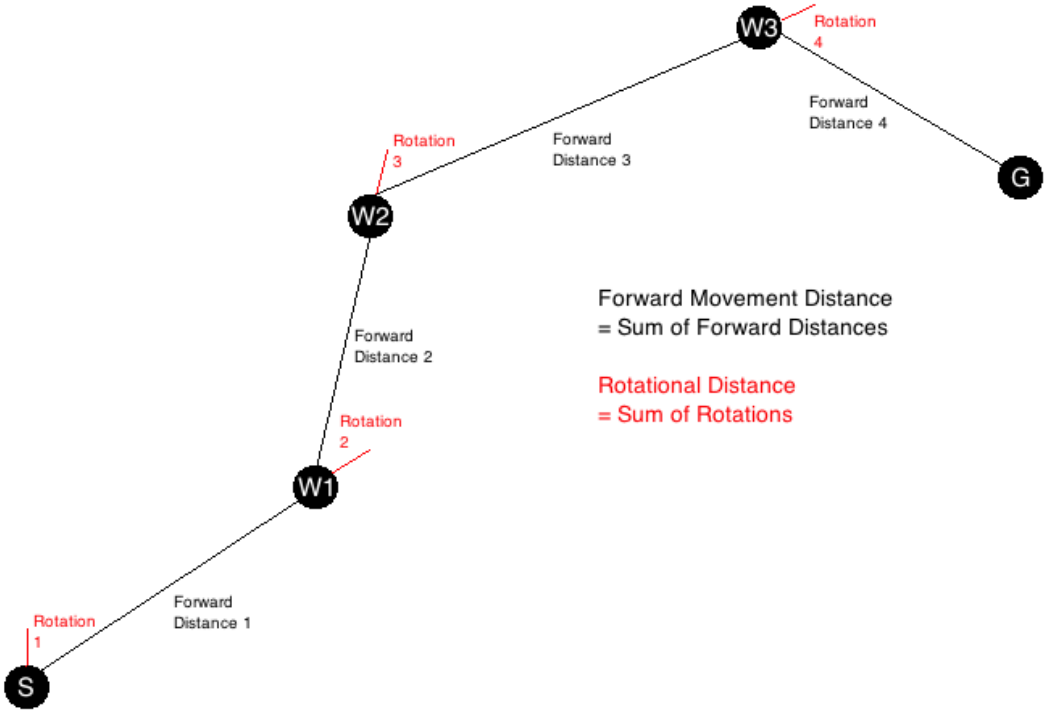


Figure 7.3: Example of distance measurements

terminated early and subsequently rerun due to the danger of Helikite damage on low-hanging obstacles in the test environment.

Vehicle loss: Unlike the fixed viewpoint pre-tests, the final testing did not experience any loss of the vehicle during execution. This is possibly due to the constrained motion of the aerial vehicle relative to the ground vehicle caused by the fixed tether and lack of air movement. The lack of air movement typically meant that the Helikite was located directly above the target vehicle due to upwards lift caused by helium.

Collision: No collisions were recorded during these tests, anecdotally the vehicle did come quite close to the obstacle on several occasions, typically due to latency in the control system causing overrunning.

Terminating distance: The test parameters were set such that if the vehicle perceived itself to be located within 10 centimetres of its current sub-goal or goal, then that goal was reached. The terminating distance is the straight line Euclidean distance between the final goal location and where the system perceives itself to be located when it terminated. Table 7.1 details the average terminating distance for each type of test. Type of obstruction appeared to make no difference to the terminating distance, the mean values for each are broadly similar with a high level of standard deviation between each test. Likewise, start to goal angle seemed to make no noticeable difference to the terminating distance, with each being fairly random though limited to 10cm. The large variation in termination distance and consequential standard deviation is likely to be attributable to the reasonably high and irregular time between updates and latency in the control system. As the vehicle turns towards the vehicle and will proceed forward if it is facing a 10cm radius around the goal, the intersection of the goal radius will be at different distances and will have to coincide with a new location update.

Final location and distance to the goal: Due to the latency between capture and locational updates, the final location and terminating location are different, caused by latency-driven overshooting of the goal before the vehicle realises that it is close enough to terminate. The final location and its associated distance are measured after termination and a sufficient period of time passing, currently set to five update cycles to allow for latency and can be considered as the final location of the ground vehicle as measured by the system itself using stitching and homography. The final locations and distances to the goal are shown in Table 7.2. Typically, the mean final distance to the goal was between 15cm and 30cm but did exceed this in a single case. Between obstruction types, there seemed to be little variation, though the mean values changed, the standard deviation was high suggesting that differences were due more to luck than any particular cause. Like the obstruction types, the start to goal angle didn't seem to make a noticeable difference to the final distance, with each being broadly similar in terms of average and standard deviation. The final distance to the goal, like the terminating distance is

Table 7.1: Terminating distance from goal

Obstruction Type	Start to goal angle (degrees)	Mean terminating distance (cm)
None	45	6.51 (1.55)
	135	7.22 (1.94)
	-135	5.44 (3.70)
	-45	6.29 (1.72)
Static	45	7.13 (2.82)
	135	6.23 (2.81)
	-135	4.86 (2.90)
	-45	6.76 (2.46)
Introduced	45	6.56 (1.59)
	135	5.23 (2.03)
	-135	6.03 (1.45)
	-45	7.85 (1.47)

seemingly random and affected by the current pose of the robot, and time between updates. Theoretically, the final location and distance to goal should be a continuation of the last straight line movement made (that took the vehicle into the termination radius), however in practice this doesn't yield much information given the variability of approach angles, distance from goals of termination and time between updates, which can lead to wildly different distances even between similar test runs. In practice, the overshooting of the goal caused by latency is likely to be more of an issue in areas where there are high level of obstructions, as the amount of overshooting doesn't really change between runs, it is more likely to be relative to the speed of the robot and the amount of latency in the control system.

Time taken: The time taken between the first update and the arrival of the robot at its goal destination are shown in Table 7.3 and Figure 7.4. All being well, it should have been expected that obstructionless runs take the least amount of time, followed by fixed obstacles and then runs with an introduced obstacle given the extra expected time taken not only to replan but also the increased movement distance required to navigate around the obstacle. As expected, the time did increase with the type of obstruction at all angles. In terms of angles, as expected the goals with a smaller angular turn both took less time to complete than those with increased rotations. In all cases, there is a large amount of deviation between times, more than likely due to the large number of variables such as update speed, Helikite motion etc. In practice, it did seem that even where runs were the same, they did vary significantly depending on the amount of overturning and overshooting caused by latency and irregular time between updates. In practice, the time did increase as expected, the rotation seems to be a significant

Table 7.2: Final distance to the goal

Obstruction Type	Start To Goal Angle (degrees)	Final Distance To Goal (cm)
None	45	23.75 (5.03)
	135	20.89 (7.79)
	-135	25.21 (9.04)
	-45	24.83 (6.25)
Static	45	16.12 (8.39)
	135	18.99 (8.01)
	-135	26.04 (8.73)
	-45	16.93 (9.35)
Introduced	45	17.54 (12.32)
	135	24.69 (6.38)
	-135	31.19 (9.82)
	-45	14.36 (8.33)

factor in the time taken and as such more convoluted paths with a larger number of waypoints are likely to take significantly longer than simpler paths.

Number of replans: After the initial five updates to the map, the initial path is planned. As the vehicle moves, it builds and updates its map, and may need to replan the path due to increased knowledge about the traversability of its surroundings. Given the relatively small map area and field of view, it would be expected that both the obstacle-free and static obstacle paths would not require replanning, and the introduced obstacle paths should only require a single replan (after the introduction of an obstacle). Table 7.4 shows the number of updates, as expected there were no updates in the static map and updates where an obstacle was introduced. Between different goal locations, there is no identifiable variation in number of replans, except in the 45-degree test, it seems that one of the planners replanned multiple times, possibly due to an unintended obstacle in its path rather than because of a difference in angle. In practice, the replanning worked as expected, planning appropriately given new data.

Forward and rotational movement distance: The sum of all forward movement between the start and goal location, and the sum of all rotations made between the start and goal location. Data is shown in Table7.5 and Figures 7.5 and 7.6. Theoretically, the forward movement distance should be lowest in obstruction free tests, followed by fixed obstruction, then introduced obstructions. In terms of start to goal angle, the 45 and -45-degree goal angles should require less turning and therefore have a lower rotational movement distance in comparison to the 135 and -135-degree runs. For forward movement, the distance was as expected shortest in the obstruction-free tests, rising in the fixed obstruction and peaking in the introduced obstruc-

Table 7.3: Execution time

Obstruction Type	Start To Goal Angle (degrees)	Time Taken To Reach Goal (seconds)
None	45	323.79 (36.43)
	135	405.46 (59.48)
	-135	373.14 (37.58)
	-45	344.96 (49.69)
Static	45	459.88 (50.12)
	135	793.02 (242.35)
	-135	792.61 (123.17)
	-45	463.42 (92.14)
Introduced	45	914.54 (211.69)
	135	813.73 (139.31)
	-135	905.60 (165.05)
	-45	606.27 (138.38)

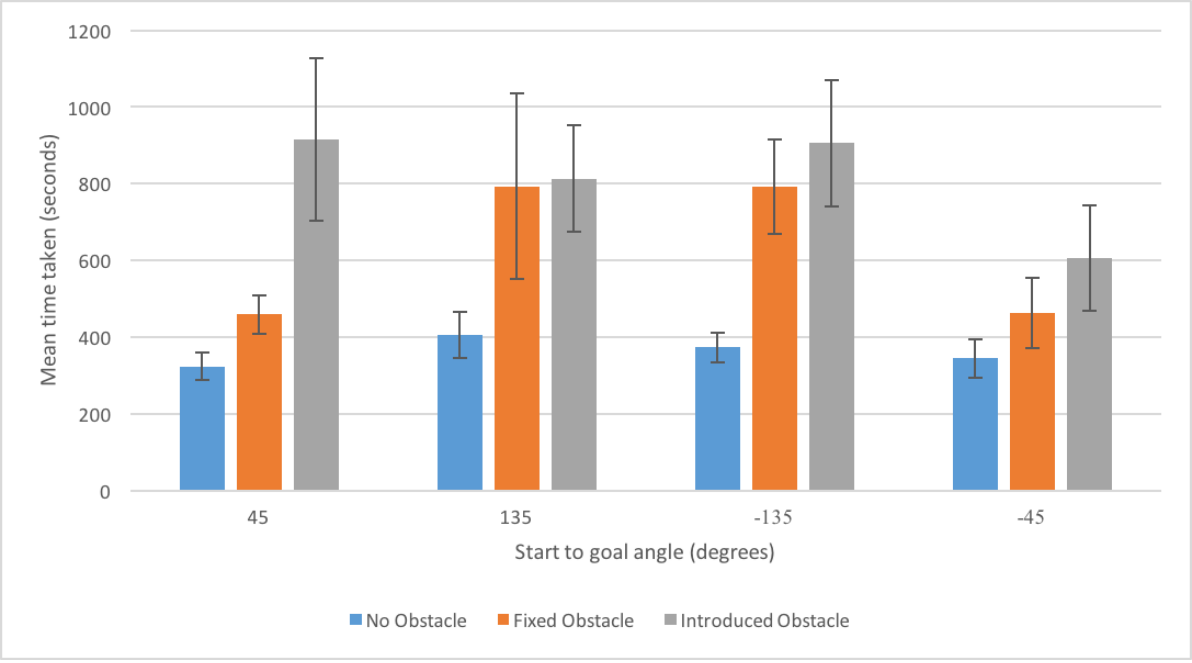


Figure 7.4: Path execution time

Table 7.4: Replanning

Obstruction Type	Start To Goal Angle (degrees)	Mean Number of Replans
None	45	0 (0)
	135	
	-135	
	-45	
Static	45	
	135	
	-135	
	-45	
Introduced	45	1.20 (0.45)
	135	1.00 (0)
	-135	1.00 (0)
	-45	1.00 (0)

tion tests. Interestingly, however, the difference between the fixed and introduced obstruction varied, possibly due to the variation in amount moved before replanning. Relative goal location did not particularly affect the forward distance in any of the tests, the robot simply rotating less or more. Rotation increased hugely with increased waypoints due to obstructions and also varied with the differing goal location. The standard deviation of rotational movement is quite high possibly due to the tendency for the vehicle to over-rotate in some runs, particularly where there are multiple waypoints which can lead to a huge variation in cumulative rotational movement. Interestingly, the 135 and -135 degree rotation with introduced obstructions had a typically lower rotational distance than its obstructionless or static obstacle counterparts, this is more than likely due blind luck than any other factor, with latency-induced overturning simply happening less on these runs. Practically speaking, the changes were as expected, though there was huge variability between tests, likely due to different planned paths, times between updates, and the introduction of obstacles at different points in the path.

Final real world location: As well as output from the system after termination and allowing five update cycles for latency to obtain the final location and distance to goal, the actual real-world location of termination was measured using a tape measure as x and y offset from the start, and straight line, Euclidean distance from the goal. Offsets as well as distances from the goal point are shown in Table 7.6, the resultant mean locations have also been plotted in Figure 7.7. The data suggests that there isn't really a discernable pattern between obstruction types or goal location types, given the similarity of the values and high standard deviation. The location, therefore, seems to be fairly random, like the other two measures (terminating distance and final location), it seems to be relative to the final movement of the robot before

Table 7.5: Vehicle movement

Obstruction Type	Start To Goal Angle (degrees)	Mean Forward Movement (cm)	Mean Rotational Movement (radians)
None	45	188.07 (9.05)	3.88 (0.51)
	135	221.18 (14.54)	5.04 (0.66)
	-135	216.43 (24.32)	4.42 (1.32)
	-45	203.46 (28.47)	4.71 (1.21)
Static	45	320.91 (34.89)	4.47 (0.09)
	135	427.17 (109.75)	9.33 (2.62)
	-135	418.74 (77.95)	9.87 (2.41)
	-45	316.72 (74.14)	5.26 (2.77)
Introduced	45	434.43 (101.20)	9.91 (3.54)
	135	438.01 (47.99)	5.91 (2.41)
	-135	422.20 (47.03)	4.14 (0.93)
	-45	397.26 (98.36)	7.04 (2.34)

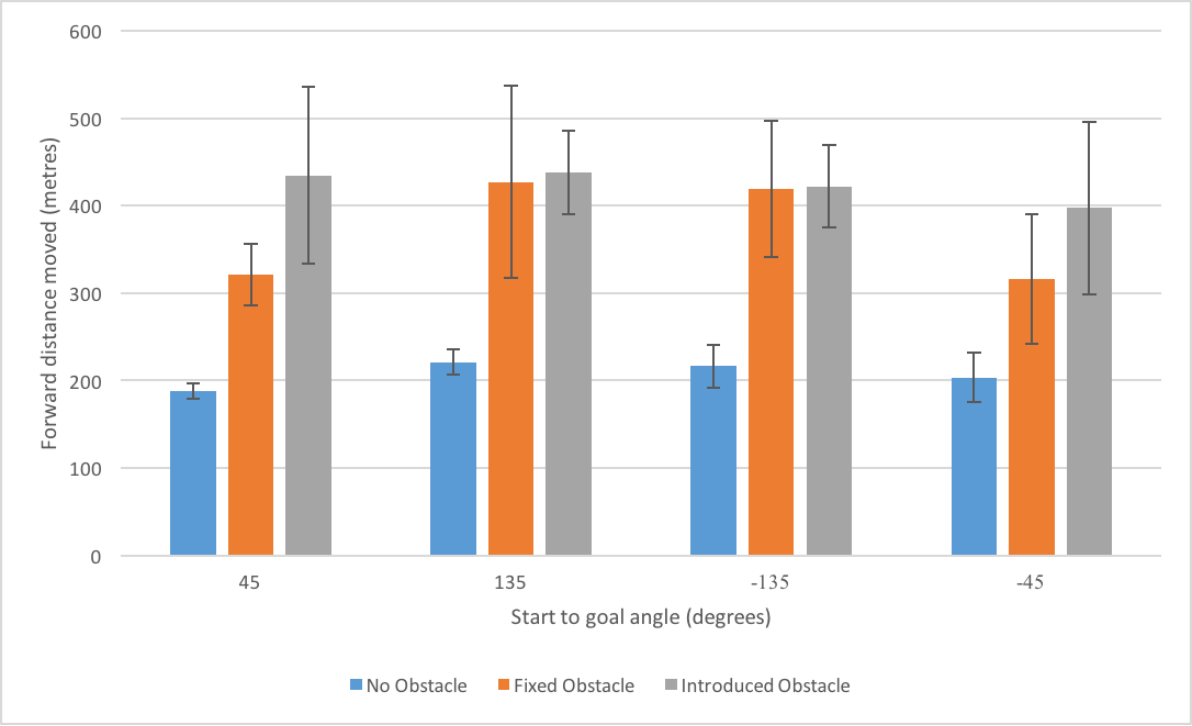


Figure 7.5: Forward distance travelled

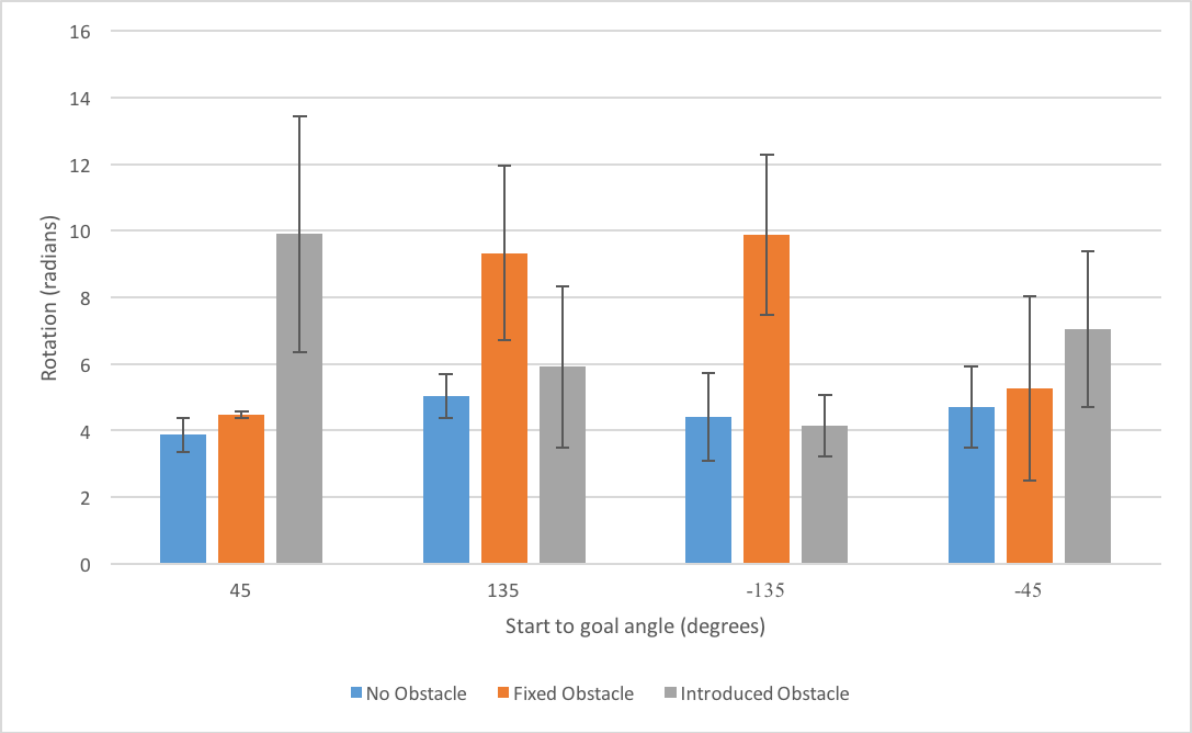


Figure 7.6: Rotational movement

termination, the location of the intersection of the termination radius, and the time between updates. Interestingly, the data differs from the calculated locations in Table 7.2 which would suggest a slight misalignment in stitching, or inaccuracy of manual measurement, which will be considered more in Section 7.4.2.

A note on differing locations: There are three different locational and distance metrics used, each of which differs. Terminating distance/location is the systems measured distance/location relative to the goal upon termination, which should be less than 10cm to allow termination. The final location is the actual location measured upon termination, as measured by the vehicle and will differ from the terminating distance/location due to the latency of the system and movement within this latency period. Final distance/location is measured by allowing sufficient time after ceasing of vehicle movement for the ground vehicle location to update, currently set at five update cycles. Real world location is a physically measured location, relative to the start location that is measured after termination. Differences between final and real-world location and distance are likely due to miss-stitching and mapping during the execution process.

Figure 7.7: Final ground vehicle locations

Table 7.6: Final real world location

Obstruction Type	Start To Goal Angle (degrees)	X offset from start (cm)	Y offset from start (cm)	Mean straight line distance from goal (cm)
None	45	131	131	43.84 (10.49)
	135	152	112	67.20 (22.46)
	-135	128	112.6	35.76 (18.31)
	-45	128.2	128.2	39.88 (16.11)
Static	45	118	118	29.23 (24.39)
	135	116	105	19.47 (2.70)
	-135	131	102	31.25 (7.80)
	-45	118	99	24.88 (17.18)
Introduced	45	126	120	38.28 (29.88)
	135	129	104	31.38 (7.12)
	-135	130	120	39.78 (6.84)
	-45	116	111	26.22 (12.11)

7.4.2 Stitch quality

Ideally, to measure the quality of stitching, a series of captured and stitched images from the system would have been compared to manually aligned images to confirm correct functionality, unfortunately, due to the large number of captured frames and time constraints, this was unfeasible. As an alternative, two measures of measuring the quality of stitching are proposed: initialisation difference and final location difference which measure the alignment of stitching at the beginning and end of runs, data is shown in Table 7.7. At the start of the runs, the first five updates are captured and added to the newly generated map before planning and movement starts, as such the position doesn't change during these update frames. Initialisation difference is the movement that takes place during the first five frames, given that the physical position won't have changed, any difference must be due to slip in the stitching process. Final location difference uses the difference between the calculated final location and the physically measured distance and should measure the slip over the course of execution.

Both measures should not vary despite changes in obstruction type or goal location. In practice, initialisation difference doesn't vary significantly between run types, given the fixed number of frames, this is not surprising and is relatively small. The final location difference is larger and has a large standard deviation which will be due to the accumulation of slight misalignments over the course of the run, and as such may increase over longer runs. Though the final location difference appears to be quite high, it is likely higher than in real-world conditions due to a featureless test space and a small field of view caused by a low flying height which means that good quality features may be lacking in the test images, it will also

Table 7.7: Stitch quality

Obstruction Type	Start To Goal Angle (degrees)	Mean Initialisation Difference (cm)	Mean Final Location Difference (cm)
None	45	3.88 (3.26)	23.86 (6.32)
	135	2.07 (1.45)	34.94 (27.02)
	-135	7.54 (3.70)	18.72 (11.94)
	-45	2.50 (1.42)	22.33 (11.68)
Static	45	5.20 (2.25)	24.27 (18.62)
	135	5.27 (2.75)	13.44 (11.34)
	-135	2.32 (1.78)	12.79 (12.82)
	-45	5.60 (4.77)	12.34 (15.09)
Introduced	45	3.15 (2.78)	36.06 (32.19)
	135	2.72 (1.88)	23.11 (19.23)
	-135	2.85 (1.96)	12.68 (7.73)
	-45	3.22 (2.47)	20.19 (17.65)

likely vary slightly between different surfaces.

7.4.3 System planning and movement accuracy

The system captures images, converts the frame to a texture based image, transmits the imagery to the GroundPi and uses this data to generate and maintain a map and current location/pose of the ground vehicle. If the assumption is made that the system performs accurately until the mapping stage, we can evaluate the difference between planned and executed paths in terms of the area between paths and the length of paths as shown in Figure 7.8. Data is shown in Table 7.8.

There is a clear difference between the planned and executed path in terms of the area between paths and path length ratio.

For area between paths, it was expected that due to the presence of a single optimal path (start to goal), the area would be low, rising where there is an obstacle, with potential paths either side. Theoretically, there should not have been a great deal of difference between start angles. In practice, the predicted rise in path difference did materialise between obstruction types. Interestingly, where the target is at 135 degrees from the vehicle, the path difference is typically higher than those tests with a 45-degree goal whether obstructionless or obstructed which may be due to increased overturning because of latency. The data seems to suggest that path difference is additive i.e. it increases with the length of the path, which makes sense considering that a longer path with multiple possible paths around obstacles will have more opportunity to deviate from the optimal planned path.

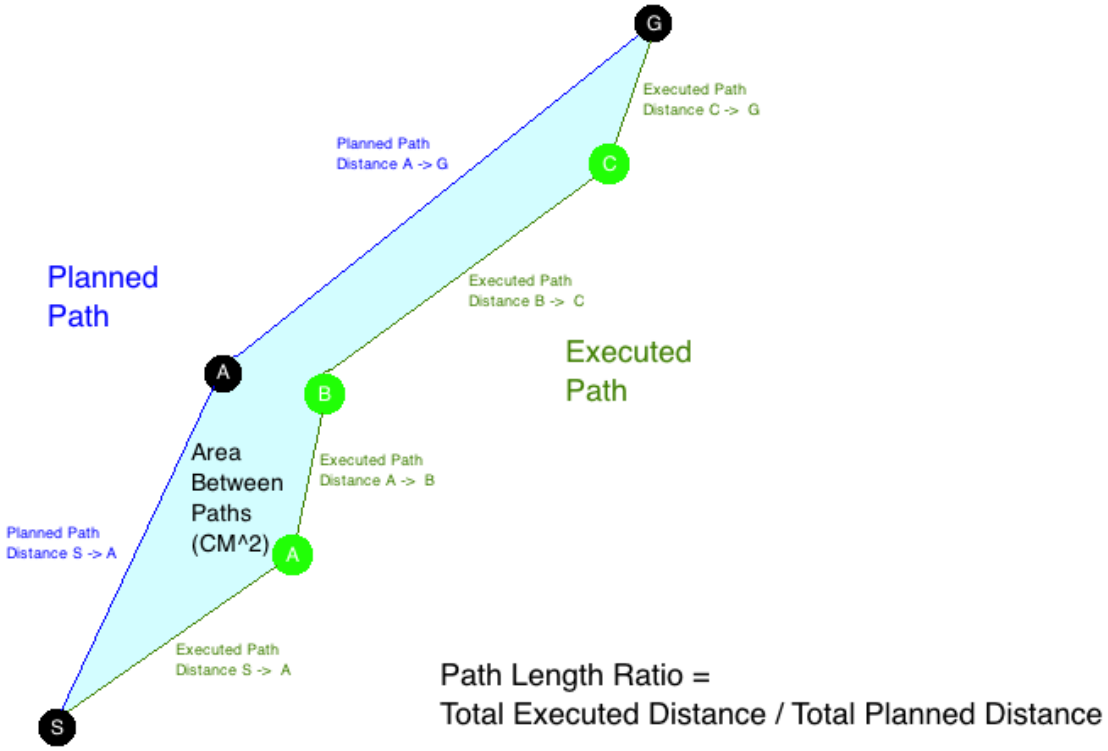


Figure 7.8: Path accuracy metrics

Table 7.8: Movement accuracy

Obstruction Type	Start To Goal Angle (degrees)	Mean Actual VS Planned Difference (CM^2)	Mean Actual / Planned Path Length Ratio
None	45	545.2 (207.17)	1.28 (0.07)
	135	753 (382.36)	1.53 (0.10)
	-135	707.4 (293.03)	1.49 (0.17)
	-45	533.4 (131.06)	1.44 (0.20)
Static	45	1267.8 (396.24)	1.53 (0.08)
	135	2186 (847.92)	1.80 (0.37)
	-135	1797 (570.96)	1.85 (0.20)
	-45	1396 (553.78)	1.41 (0.04)
Introduced	45	2208.2 (726.74)	1.84 (0.53)
	135	2932.1 (1425.8)	2.19 (0.27)
	-135	3165.1 (1601.73)	2.30 (0.19)
	-45	2157.6 (564.16)	1.74 (0.32)

Path length ratio in a perfect world should be relatively constant, however, given the overturning of the vehicle during navigation, I would suspect that longer paths or those with more turns would have a higher path length ratio. In practice, the ratio of planned to executed path distance is similar in that it increases with the type of obstruction and rotation but at a smaller rate. In real-world terms, the increase in the area between paths is fairly inconsequential, as the area between paths is not related to execution time or cost, and as mentioned in section 7.4.4, is likely to vary based on how the planner decided to navigate around an object. The increase in path distance ratio with path complexity, is again not surprising given that more complicated paths are likely to have increased turns with the potential overturns. The consequence of an increased path length ratio, is clearly not fantastic, as path complexity increases the data seems to suggest that the vehicle will become less efficient, taking more time to reach the goal. In real-world scenarios, I would expect any increase in path length caused by overturning to be offset by longer straight-line sections of path that would reduce the overall distance ratio.

7.4.4 Planned and executed path optimality

The RRT planner is not guaranteed to be optimal or complete in comparison to Dijkstra, A* or D* planners, but has advantages in terms of speed. In cases where there are no obstacles, the most efficient and shortest path is a straight line between start and goal locations. Where there are obstacles, however, the shortest path is likely to have to avoid obstacles minimally, that gets as close as possible without collision. To evaluate path optimality, the planned RRT path was compared with an optimal path generated by the A* planner.

Whilst using the difference in area between the non-optimal RRT and A* could have been used as above, it was discounted in favour of path distance. Where there are no obstructions, the area path difference is likely to be minimal or none. However, the problem with path difference is that where there are obstacles and multiple different possible paths, the area will vary hugely if the path chosen is not geographically similar to the optimal A* path but may differ little in terms of path length. An example of a scenario where path difference fails is where a planned route traverses past the left of an obstacle whereas the optimal route passes to the right causing a large gap between the paths.

Table 7.9 and Figures 7.9 and 7.10 show both the area between paths and the distance path difference. As explained where there are no obstructions, the path difference is small (approximately 1000cm^2 / 0.1m^2), confirming that the RRT planned path is nearing towards optimal, however, this rises disproportionately with obstacles, possibly due to there being many possible paths. For the distance ratio of the RRT planned and A* optimal paths, in most cases, the planned paths tend to be very close to optimal (approximately 1 to 1.5 times the optimal distance). In some cases, the planned distance is less than the A* distance, this can be

Table 7.9: Planned path optimality

Obstruction Type	Start To Goal Angle (degrees)	Mean A* VS Planned Path Difference (CM^2)	Mean Planned / A* Path Length Ratio
None	45	1113.8 (558.44)	0.99 (0.07)
	135	1039.6 (252.66)	0.95 (0.02)
	-135	1008.4 (443.40)	0.97 (0.04)
	-45	781.2 (345.71)	0.95 (0.01)
Static	45	4654.40 (3307.87)	1.15 (0.11)
	135	5607.60 (4675.07)	1.32 (0.19)
	-135	4140.20 (3951.45)	1.27 (0.22)
	-45	5832.20 (5427.01)	1.20 (0.23)
Introduced	45	7604 (3941.97)	1.38 (0.17)
	135	7542.00 (3638.59)	1.12 (0.12)
	-135	4042.40 (2636.64)	1.01 (0.01)
	-45	6414 (4279.99)	1.27 (0.22)

explained by the ability of RRT to plan at any angle. Although paths with obstacles typically were not optimal in comparison to A*, anecdotally they had far fewer waypoints which would be more suitable for real-world traversal as turning caused by waypoints seemed to increase execution time significantly in comparison to straight line movement.

Although the planned paths are close to optimal, the real world travelled paths are likely to be less close to optimal because of various factors such as latency in measurement and accuracy of sensors. Results shown in Table 7.10 demonstrate the difference between a theoretical A* (optimal) path and the travelled paths using both path difference and ratio of executed path length as metrics.

Despite the latency of the system, the ratio of the path length to the optimal A* path length never exceeds 2.5 with lower ratios where there are no obstacles present. Where there are obstacles, this ratio rises likely due to the increased number of turns and potential for overshooting caused by the latency. The area between paths or path difference is not particularly helpful as a measure where there are obstacles because of the possibility that paths could be planned that are similarly optimal but geographically very different. Where there were no obstacles, the path difference was quite low (~1000CM^2), however, this rises where there are obstacles, potentially indicating a large geographical difference between the A* and executed path. In comparison to the planned path, the executed path is more different both in terms of the area between paths and distance ratio.

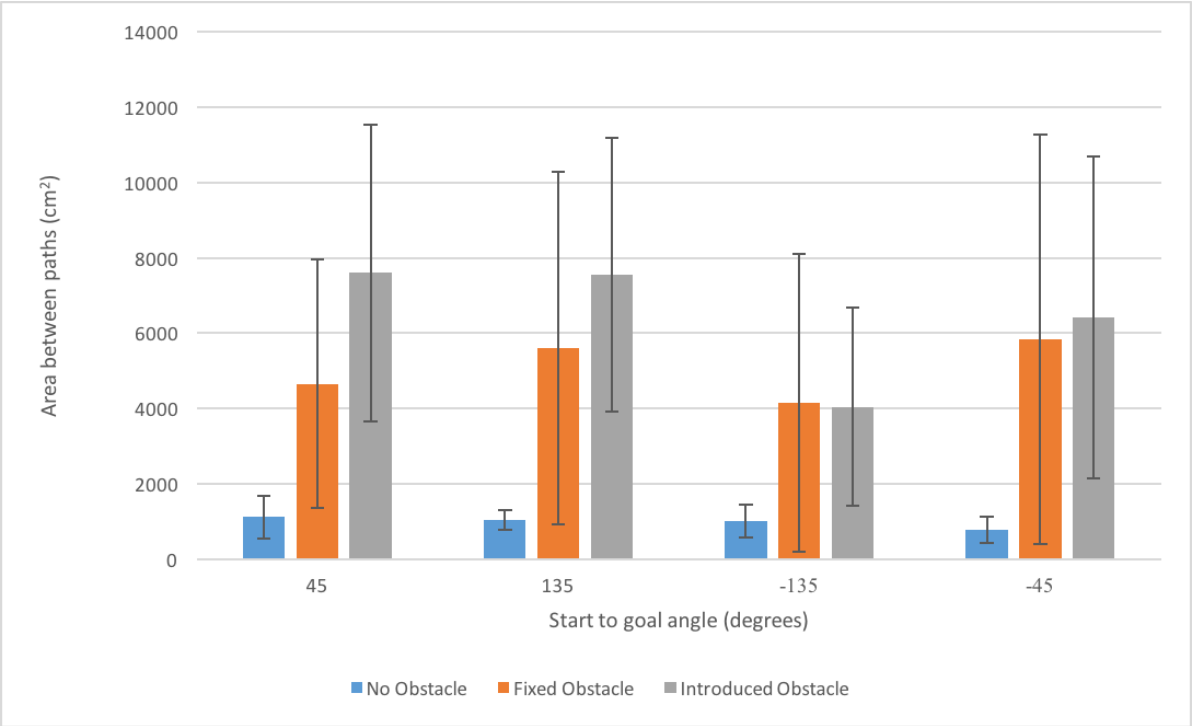


Figure 7.9: Executed area between path optimality (RRT versus A*)

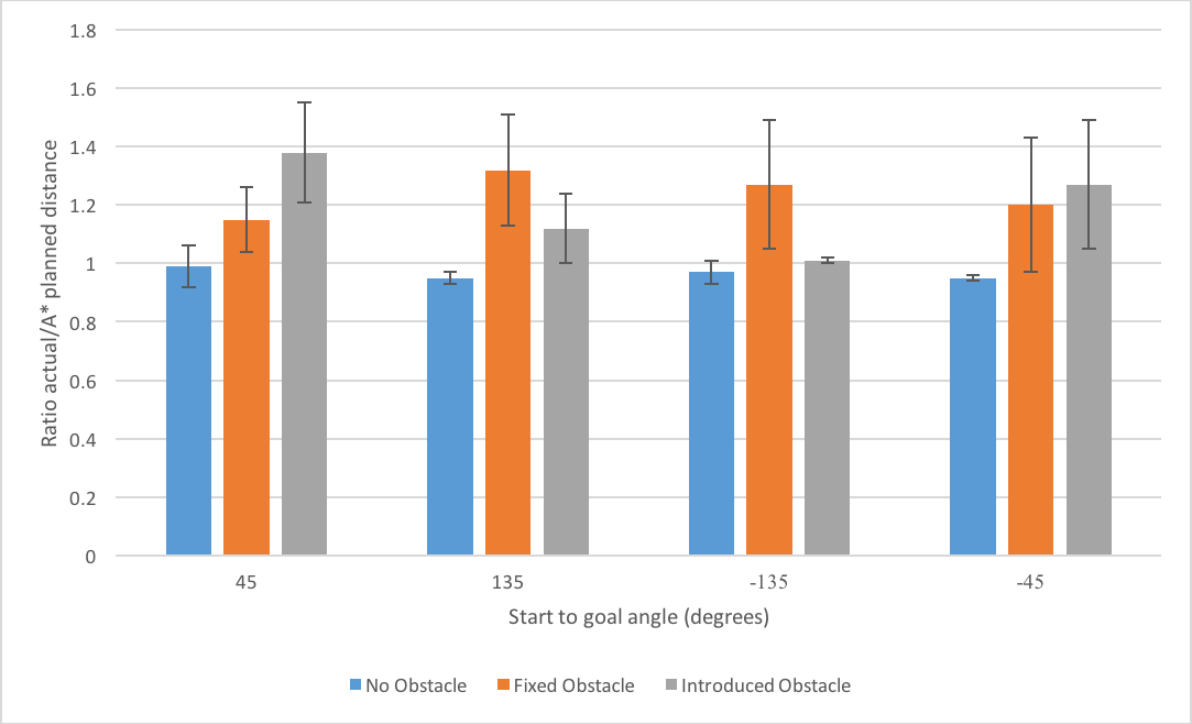


Figure 7.10: Executed path distance optimality (RRT versus A*)

Table 7.10: Executed path optimality

Obstruction Type	Start To Goal Angle (degrees)	Mean Actual VS A* Path Difference (cm^2)	Mean Actual / A* Path Length Ratio
None	45	1084.20 (427.66)	1.26 (0.08)
	135	1206.60 (184.13)	1.46 (0.09)
	-135	1525.40 (357.21)	1.45 (0.17)
	-45	987.40 (449.40)	1.37 (0.20)
Static	45	5365.80 (3421.38)	1.76 (0.24)
	135	6745.60 (4895.15)	2.39 (0.64)
	-135	4236.00 (3373.09)	2.34 (0.45)
	-45	6343.40 (5864.60)	1.70 (0.38)
Introduced	45	9276.60 (4391.32)	2.50 (0.56)
	135	10987.00 (1150.74)	2.45 (0.29)
	-135	10833.20 (894.19)	2.32 (0.18)
	-45	8048.40 (4224.65)	2.24 (0.65)

7.4.5 System speed

The speed of the system as well as being measured in terms of time between updates can be measured as the time taken to traverse the path between start and goal. Table 7.11 attempts to estimate the average movement speed of the combined system using the knowledge about the path length, time taken and amount of time spent actually moving. The average speed is then used to estimate the system lag in Table 7.12. The forward movement speed didn't really vary between obstruction types or goal locations, tending to be between 0.4 and 0.7cm/s, it doesn't, however, take into consideration time spent rotating which means the actual forward speed is likely to be significantly higher. System lag was calculated to be between 21 and 70 seconds. The theoretical estimates of 21 to 70 seconds lag appear to be very high in comparison to the actual lag experienced which may be due to rotational time being considered when calculating forward speed. In practice, the system latency is likely caused by a combination of factors, primarily the high cost of texture conversion and transmission to the ground vehicle.

7.5 Indoor constrained path testing

The main series of tests evaluated the performance of the system in terms of its ability to avoid a single obstacle in an empty area but didn't evaluate its performance where there is only a single path. To evaluate performance in an environment with more obstacles or a single valid path, paths were planned in a constrained environment. Similar to the main series of tests, runs

Table 7.11: Forward movement speed

Obstruction Type	Start To Goal Angle (degrees)	Mean Forward Movement (cm)	Time Taken (seconds)	Time Spent Moving (seconds)	Average Speed (cm/s)
None	45	188.07	323.79	225.00	0.58
	135	221.18	405.46	261.00	0.55
	-135	216.43	373.14	234.00	0.58
	-45	203.46	344.96	260.00	0.59
Static	45	320.91	459.88	262.00	0.70
	135	427.17	793.02	444.00	0.54
	-135	418.74	792.61	464.00	0.53
	-45	316.72	463.42	284.00	0.68
Introduced	45	434.43	914.54	442.00	0.48
	135	438.01	813.73	439.00	0.54
	-135	422.20	905.60	572.00	0.47
	-45	397.26	606.27	356.00	0.66

Table 7.12: Latency estimation

Obstruction Type	Start To Goal Angle (degrees)	Average Speed (cm/s)	Distance between terminating and final destination (cm)	Estimated Lag (seconds)
None	45	0.58	24.46	42.11
	135	0.55	24.16	44.28
	-135	0.58	28.69	49.46
	-45	0.59	26.67	45.22
Static	45	0.70	18.47	26.48
	135	0.54	16.41	30.46
	-135	0.53	24.04	45.50
	-45	0.68	19.32	28.27
Introduced	45	0.48	17.90	37.68
	135	0.54	24.89	46.24
	-135	0.47	32.15	68.97
	-45	0.66	14.20	21.68

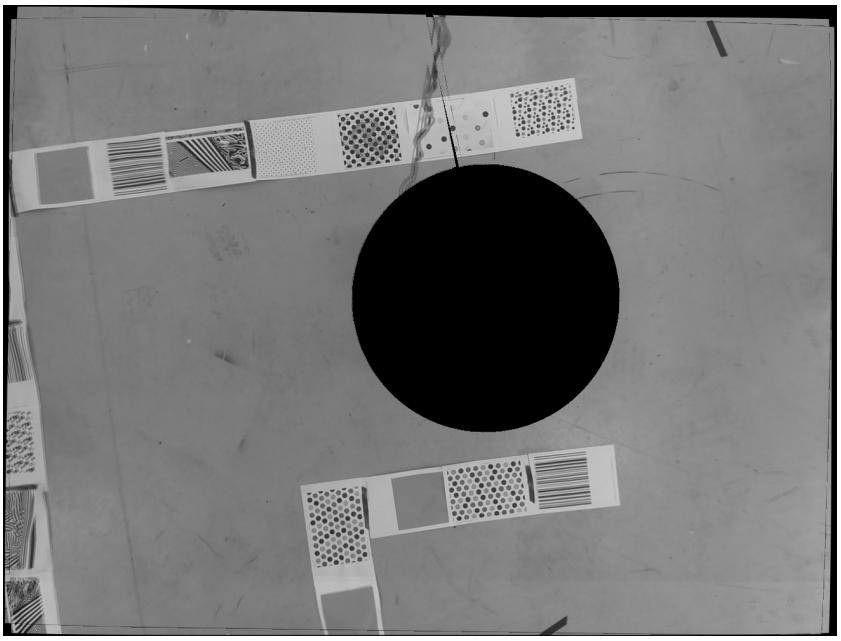


Figure 7.11: Constrained path example image

were made in a single orientated L shaped path at a distance of 100cm forward and 100cm left of the front of the robot from the pose and orientation of the vehicle at the start of the run. This type of test was repeated five times to simply confirm correct functioning of the system (see Figure 7.11 for an example image).

Similar to the static obstacle tests, the constrained path runs moved in an L shape goal from start to goal. Like the main series of tests, the goals were reached without collision, though due to the latency the vehicle did get quite close. The constrained nature of the map (See Figure 7.12) did mean that the path was more optimal, due to the inability of the RRT planner to plan lengthy paths, instead being constrained.

7.6 Outdoor testing

After completing both motor control tuning and indoor testing in Chapter 5 and above, the existing movement systems were combined in a series of outdoor tests to evaluate the performance of the system as a whole. The outdoor tests were performed across different distances and on multiple surfaces in order to evaluate real-world performance of the combined system and traversal of longer distances with powered aerial vehicle movement. Two locations were evaluated, example views of which are shown in Figures 7.13a and 7.13b for grass and 7.13c and 7.13d for tarmac. Additionally, a paved area was planned to be evaluated in order to evaluate performance on a more variable surface, as well as some real-life constrained path

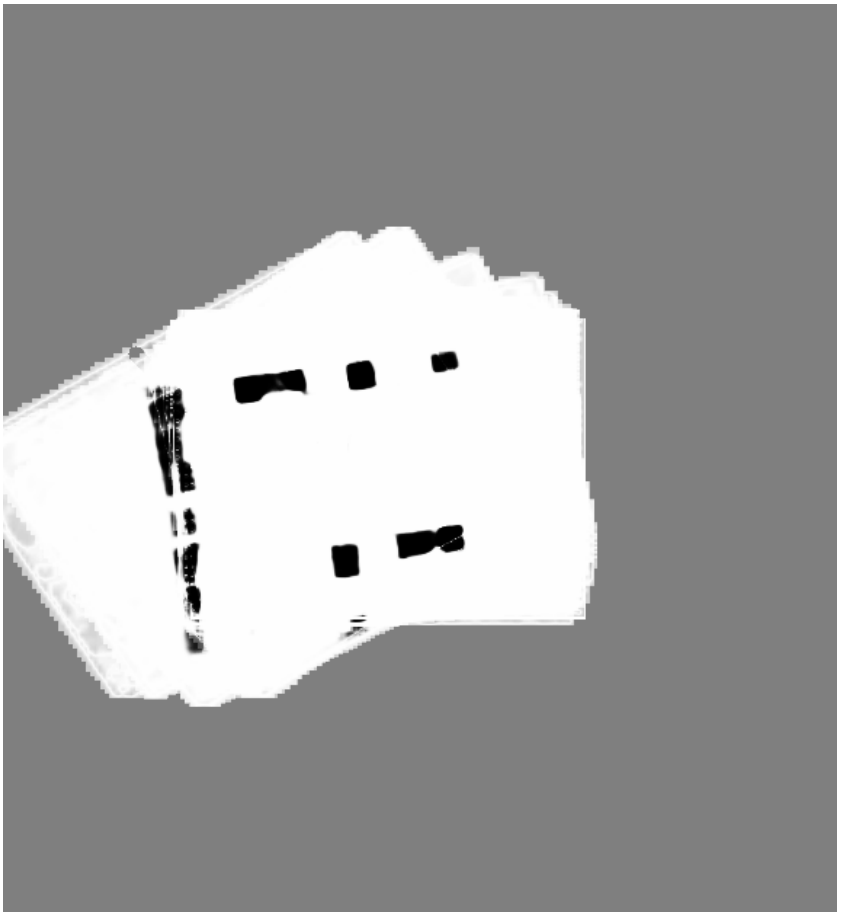
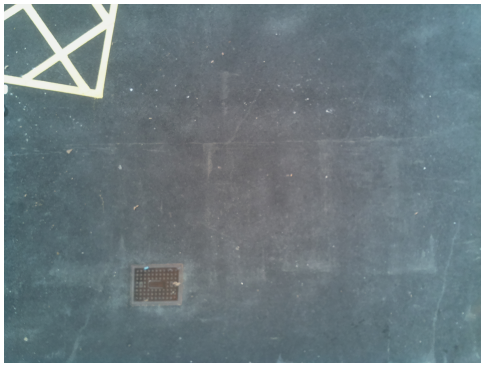


Figure 7.12: Constrained path map

(a) Grass test location

(b) Grass overhead view

(c) Tarmac test location



(d) Tarmac overhead view

Figure 7.13: Outdoor location images

testing (e.g. following a path/footpath) but these were discounted after the performance of the grass/tarmac testing.

7.6.1 Outdoor test parameters

In the main, the testing followed the parameters of the indoor testing by attempting to proceed from a start to a goal location. Some alterations were made to the test parameters given the new environments, and to facilitate the addition of the motor:

- Locations(s): In contrast to the indoor testing which used a grey concrete floor with little or no pre-existing markings, two additional outdoor surfaces were proposed grass and tarmac which were both unknown to the software. Whereas the indoor test had a uniform colour, the outdoor test environments are often less uniform in both colour and therefore texture, are more susceptible to changes in natural light and may have other unplanned objects in view such as leaves, cars or road markings.
- Repetition: As with previous tests, each test was intended to be repeated five times.

Given that there were two locations, three distances and three different types of obstruction this meant that ninety runs took place.

- Rotation: The indoor testing varied the rotation throughout the experiment from start to goal which demonstrated that increases in the amount of rotation extended the distance travelled and the time taken significantly. Given that rotation has already been evaluated and that the ground vehicle movement algorithm has not been altered, tests were performed in a single direction i.e. without altering the rotation between tests in favour of altering the distance instead.

- Distance: Previous tests used a single distance between start and goal which was the 1,1m in the x and y direction from the start location. For this series of tests, the distance travelled was altered to three different path lengths of two, five and ten metres, in order to evaluate the performance over different path lengths.

- Obstruction(s): As with previous indoor testing, three types of obstruction were used. Some of the tests used no obstruction, that is there was no intended obstruction between the start and goal location. Others had an obstruction visible in the frame from the start of running whereas some had an obstruction that was introduced during runtime. It is important to note that although the obstructionless tests did not have any artificial obstructions, the nature of the outside environment may mean that some are identified by the system.

- Height: The inside tests were constrained in that the low roof and lighting limited the flying height of the balloon. For these tests, a constant string length of seven metres was used which assuming optimal lift should allow for a fairly consistent field of view.

7.6.2 Results of outdoor testing

Initially, after reactivating the motor, some initial testing took place outside to confirm that the system was functional before running the main series of tests, at this point it was immediately clear that the motor system was struggling to keep the Helikite located vertically above the target, as such some small alterations were made to the parameters of the control system to compensate:

- Maximum motor power: The maximum motor power was increased, previously this had been limited to avoid over-depleting the FlyPi cells.

- Maximum initialisation time: As part of the start process, a maximum initialisation time is set that limits the number of requests made for the first five update images from the

GroundPi to the FlyPi, if five frames aren't successfully received in this time then the software will terminate. Whilst the value used in the indoor testing was sufficient, it was clear that many of the trial runs were not starting due to a failure to initialise.

After changing the parameters, the outdoor tests were attempted. Unfortunately, of the ninety attempted runs, only eight successfully managed to complete, the distributions of which are shown in Table 7.13.

Analysis of the data shows that the runs failed for a variety of reasons, primarily caused by the lack of motor power:

- Initialisation failure: At the beginning of runs, the GroundPi unit requests a sequence of five update images from the FlyPi, in many cases these five update images (which need to have the ground vehicle in view) failed to be captured within the extended timeframe due to the ground vehicle being out of view and as such the run failed to start.
- Vehicle loss during execution: Since the current location and pose are calculated using a continuous sequence of stitched images that require the target in-frame, the lack of target in view and erratic movement of the Helikite in windy conditions (due to the lack of motor power) meant that often it was not possible to stitch images causing the vehicle to became stuck for significant periods of time and then requiring the run to be terminated.
- Motor failure: A limitation of the FlyPi system is that power consumption of the motor has to be carefully managed to prevent issues. The motor itself was selected because of its power and weight as part of the FlyPi, and is powered by two lithium polymer cells supported by the power tether. Unfortunately, during the previous testing, it was noted that sustained high motor power could quickly deplete the lithium cells more quickly than they could be replenished by the tether and as such, it was possible for the ESC to reset, ultimately causing loss of motor function. In order to prevent the motor resetting, a limit was placed on maximum motor power. Because of the increased air movement, and increased power requirements, some runs failed due to the motor resetting.
- Fail to plan: Although update images were received, an initial path was unplannable.

As briefly mentioned during the evaluation of the motor power-system, it was anticipated that the combination of the air movement due both to wind and vehicle movement could be sufficient to overpower the Helikite motor, unfortunately during the outdoor testing, it quickly became apparent that this was happening in many of these cases, with few of the runs successfully completing. Those which did not fail due to the motor typically failed because of an inability to plan a path. An overview of the reasons for failure is shown in Table 7.14.

Table 7.13: Outdoor testing successful runs

	Grass			Tarmac		
	2m path length	**5m path length**	**10m path length**	**2m path length**	**5m path length**	**10m path length**
No Obstruction	1	1	0	3	0	0
Fixed Obstruction	1	0	0	1	1	0
Introduced Obstruction	0	0	0	0	0	0

Table 7.14: Outdoor testing failed runs

Reason for failure	**Obstruction Type**	Grass			Tarmac		
		2m path length	**5m path length**	**10m path length**	**2m path length**	**5m path length**	**10m path length**
Initialisation failure	**No Obstruction**	3	2	2	1	1	3
	Fixed Obstruction	2	1	2	2	3	1
	Introduced Obstruction	2	1	2	1	1	2
Vehicle loss	**No Obstruction**	0	0	1	0	1	1
	Fixed Obstruction	1	1	2	0	0	1
	Introduced Obstruction	1	2	1	2	0	1
Motor failure	**No Obstruction**	0	0	1	0	2	0
	Fixed Obstruction	0	1	1	0	0	3
	Introduced Obstruction	1	1	1	0	4	1
Fail to plan	**No Obstruction**	1	2	1	1	1	1
	Fixed Obstruction	1	2	0	2	1	0
	Introduced Obstruction	1	1	1	2	0	1

Table(s) 7.13 and 7.14 suggest that the distribution of success and failure amongst tests is reasonably random, in that it is more likely due to other factors such as wind speed and direction than the test parameters. It seems that tests were more likely to complete at shorter distances, possibly due to a decreased likelihood of adverse gusts or winds over the shorter execution time.

Of the tests that completed, the performance of the ground robot seemed to be similar to the indoor tests, in that it turned, and traversed with an element of overturning caused by latency. Unfortunately, execution time was significantly longer, because of a significantly larger average time between updates, and paths appeared to be more convoluted. The distribution of successful runs makes it difficult to draw any particular conclusions regarding differing performance over materials, and at different from the little successfully recorded data other than subjective observations.

7.6.2.1 Hardware causes

In the majority of cases, the main cause of the failure to complete path traversal was the inability of the current equipment to generate sufficient motor power to keep and position the Helikite in typical outdoor, gusty conditions. Initialisation failure typically took place because the target was either out of view or could not be captured in enough consecutive frames to allow stitching and updating of the map. Likewise, the vehicle was sometimes lost during execution because the Helikite was located too far away from the ground vehicle to have the ground vehicle in view. Attempts to increase motor power ultimately caused motor resetting and subsequent test failure in some cases. In addition to the lack of motor power, the relatively shallow field of view of the camera hardware limited the view of the ground vehicle from the Helikite at a distance.

7.6.2.2 Software causes

In addition to the hardware issues, some features of the software implementation appeared to limit the ability of the combined system to function during the outdoor testing.

As implemented, the stitcher requires two consecutive frames, both with target detection in order to produce a valid update. Stitched images (map) were coupled to target detections in order to ensure the freshness of data and to make the most of transmission, given the cost in terms of time. Unfortunately, coupling map updates to detections meant that a lack of target view resulted in a lack of successful updates. The requirement for stitched frames to be consecutive was originally chosen to avoid the situation where two parts of a stitched image may have been taken over a long time period, instead ensuring that the two components that

made up the stitched and transmitted image were fresh. In practice, the erratic movement of the Helikite (due to a lack of available motor power) meant that these were not often captured consecutively resulting in a lack of updates.

A side effect of the addition of the motor controller and the use of natural images with more features was a noticeable increase in time between updates. The stitcher performs linearly e.g. each captured frame is stitched to the previous and therefore requires an overlap between frames. During indoor testing, images tended to overlap because of the constrained Helikite motion. In outdoor testing, the slower update speed coupled with the much faster and more erratic Helikite motion meant that consecutive captures sometimes didn't overlap, causing a failure to stitch and update. Whilst it would have potentially been possible to store and search matches such that the stitcher worked non-linearly, this would likely have decreased update speed and would require the selection of an appropriate method of storing, searching and selecting matches from multiple frames.

7.6.3 Discussion

Despite the promising performance in both the outdoor motor tests and indoor flight tests, the outdoor tests performed poorly and would require extensive hardware modification in order to be successfully completed. The root cause of most of the failures was the insufficient power of the testing platform to maintain a consistent view of the ground vehicle between frames. Where the system failed to find a path, there is some evidence that this was caused by the presence of small areas of non-traversability, which is further discussed in the alternative outdoor tests. The structure of the software, particularly the stitcher and mapper also contributed, particularly the requirement for consecutive frames for stitching and updating, which were compounded by the lack of power and relatively slow update speed in windy conditions, causing erratic inter-frame movement. In order to improve the performance of this particular test platform, extensive modifications would likely be required to the hardware. Suggested modifications to the hardware would include

- Upgrading the Raspberry Pi microcontroller to the updated Raspberry Pi 3B+, or alternative board which should lead to significantly faster updating of the map and location, with reduced latency due to its faster CPU and faster wireless connectivity.

- Upgrading the Raspberry Pi camera, to the higher resolution V2 version, offering a larger field of view and higher resolution, which in itself should allow for increased performance at height.

- Upgrading the power tether such that it is capable of carrying more current. Though an

increase in weight would affect the maximum flying height, the maximum current tether length is rarely if ever used due to the limited quality of images at that height.

- Replacing the motor and ESC with an EDF (electronic ducted fan) that is more efficient at producing forward movement for the given power.

The addition of these upgrades, would theoretically not just reduce latency but also increase movement efficiency. In terms of software, improvements could be made to remove the need for matches from consecutive frames, and to remove detections in consecutive frames for a stitch. Removing the constraint that each image is matched with the previous image (linear matching) is feasible but would be more costly in time spent searching for matches through all previously detected images, with time take increasing as more images are captured (and the database of matched points grows). Removing the requirement for stitches to be consecutive images would be simple enough to implement but would require some kind of limit on the age of captured frames used to stitch to ensure that the updates were sufficiently fresh.

Whilst the current test hardware clearly wasn't powerful enough for use in these gusty conditions, the completion of some of the runs does show that it provided a proof of concept for the use of cooperative air-ground robotics, with a novel platform and similarity measure to traverse unknown areas, and could be used as the basis for future work.

7.7 Alternative outdoor testing

The goal of the outdoor testing in Section 7.6 was to evaluate the performance of the combined system in outdoor environments. Unfortunately, the lack of motor power and failure to plan in some cases limited the functioning of the system and the amount of meaningful data gathered since most runs didn't manage to complete and would require significant modification to hardware and software in order to do so. The differences between previously performed indoor tests can be summarised as follows:

- Ground material
- Height
- Distance travelled
- Following of the ground vehicle by the aerial vehicle

Since the following of the ground vehicle and the distance travelled are linked to the satisfactory performance of the motor, these were unable to be evaluated. However, using aerial

imagery and offline processing, some of the effects of changes to height and ground surface could be evaluated, with conclusions drawn regarding the effects that this could have on a re-configured real-world system. To enable testing, a series of images were captured on multiple materials and at multiple heights using the Helikite.

- Ground materials: Like the previous outdoor tests, grass and tarmac images were collected. In addition to grass and tarmac, aerial images of a paved area were captured in order to evaluate the performance on a slightly less uniform ground medium.
- Heights: Images were captured at three, five and seven metres from the ground.
- Number of frames: Approximately 500 frames were captured for each height and ground material, some containing a target and some without.

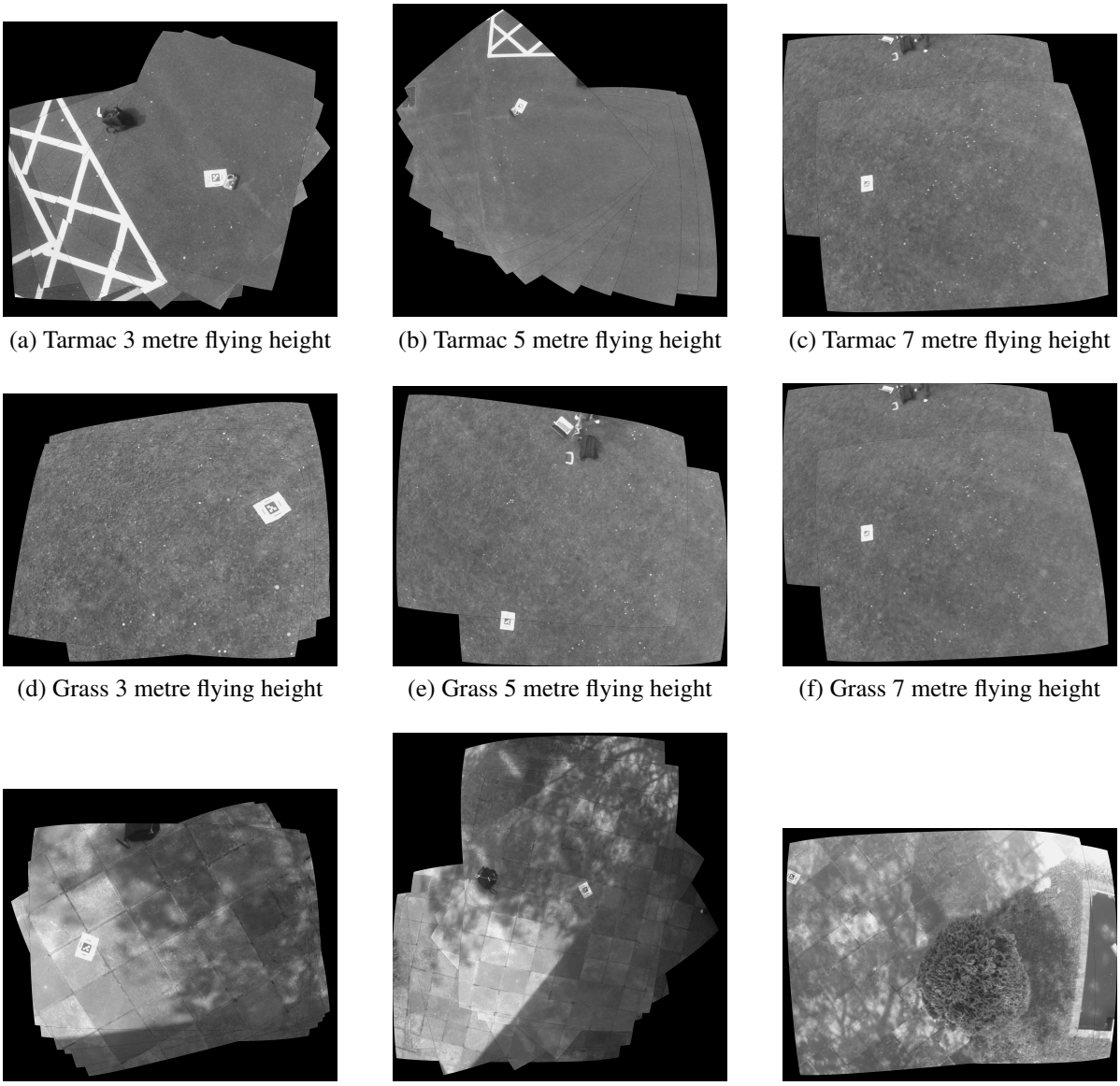
7.7.1 Ground traversal tests

In order to test the suitability of the similarity based mapper and planner on different ground materials and at different heights, a series of captured frames were stitched at each height/material combination. Similar to the threshold evaluation testing in Chapter 6, the images were then converted to a texture based similarity map and route planning was attempted. After attempts at planning, conclusions were drawn about the suitability of our texture-based similarity map on varying surfaces. The stitched images used are shown in Figure 7.14.

7.7.1.1 Safe area traversal

To confirm correct functioning on similar materials, objectively safe routes were planned i.e. between two points on the same surface without any noticeable obstruction. Should the planner function correctly, then a path should be found in all cases, which should be close to optimal, in terms of path length. For each image, three safe straight-line paths were manually selected that covered different areas of the input image, with each being repeated five times (due to the sampling-based nature of the RRT planner). Prior to the tests, it was expected that paths should be planned in most cases, similar to the previous threshold testing. Results of these tests are shown in Table 7.15.

Unfortunately, contrary to expectation, the tests were not all successful. As expected, due to its relatively uniform colour and surface, tarmac performed best on average with 68.89% of paths being planned in comparison to grass or paving which are much less uniform. The imagery of grass was typically similar in colour but being a natural material was less uniform, often featuring areas of leaf fall, changes in blade density and height, and bare mud which can impact the performance of the similarity based measure. Paving performed the worst, likely



(a) Tarmac 3 metre flying height (b) Tarmac 5 metre flying height (c) Tarmac 7 metre flying height

(d) Grass 3 metre flying height (e) Grass 5 metre flying height (f) Grass 7 metre flying height

(g) Paving 3 metre flying height (h) Paving 5 metre flying height (i) Paving 7 metre flying height

Figure 7.14: Ground traversal images

Table 7.15: Correctly planned safe paths (%)

	Tarmac	**Grass**	**Paving**	**Average**
3m flying height	66.67	0	0	22.22
5m flying height	100	33.33	33.3	55.56
7m flying height	40	33.33	26.67	33.33
Average	68.89	22.22	20	

Table 7.16: Correctly planned path distance ratio (actual planned distance / optimal distance)

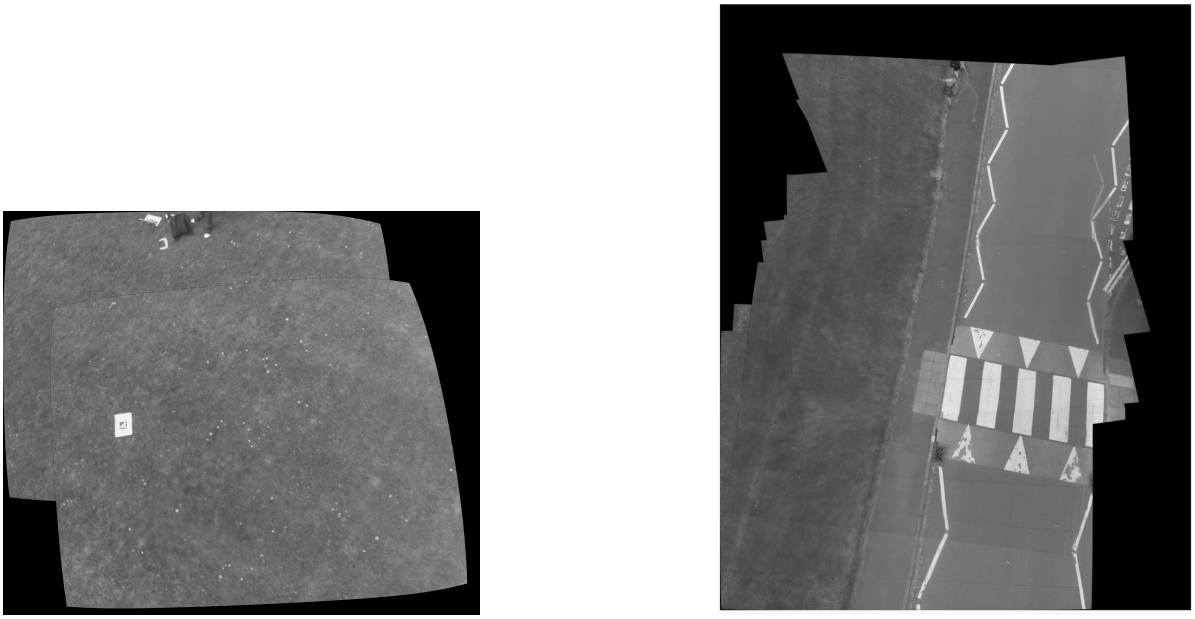
	Tarmac	**Grass**	**Paving**	**Average**
3m flying height	1.05	N/A	N/A	1.05
5m flying height	1.08	1.00	1.02	1.03
7m flying height	1.17	1.78	2.16	1.70
Average	1.10	1.39	1.59	

due to the huge variation in colour and surface pattern of the material as well as the gaps between slabs causing disparity in texture value.

At different heights, the five metre image height clearly performed most efficiently, followed by seven and three metres. Put simply, at too low a height such as three metres, the resolution highlighted any anomalous features or difference in the same material, which were subsequently reflected in the similarity based map. Conversely, at too high a height, distinct image features which can be useful for identifying unique patterns or textures become non-visible due to the camera resolution.

As expected, tarmac performed best on average with 68.89% of paths planned. In comparison to grass and paving-based imagery, the tarmac ground surface tended to be more uniform than grass, or cobble resulting in a more similar texture value. Grass imagery was typically a uniform colour, but being a natural material had a less uniform surface pattern, and often featured areas of leaf-fall, changes in density, blade height or areas of bare mud which impacted the performance of the stitcher. Paving performed the worst, probably due to the huge variability of paving slabs in terms of colour, surface and the gaps between them causing disparity in texture value and subsequent difficulties in identifying similar materials. Table 7.16 shows the path distance ratio (actual path length/optimal path length) for each of the tests. From the data, it is clear that path distance ratio mirrored Table 7.15 in that paths planned on tarmac were closest to optimal in terms of path distance, with degraded performance for both grass and paving surfaces. Unlike the previous Table, the three metre paths were on average closer to optimal than seven metre paths suggesting that although a path is less likely to be found at a three metre height than a seven metre height, if it is, it will likely be more optimal.

Interestingly, the results of these tests differ from those in Chapter 6. In all types of test, the flying height was typically lower than the images used for the threshold testing which will increase the view of smaller objects in the scene, and the ability of the texture generation algorithm to identify ground surface features. As shown in Figure(s) s 7.15a and 7.15b, the grass based imagery differed not only in height but also in the presence of small white areas caused by small flowers, leaves or petals on the ground surface. The presence of small contrasting leaves/petals on the ground surface led to small areas of non-traversability throughout the map



(a) Grass 7 metre flying height

(b) Original threshold image

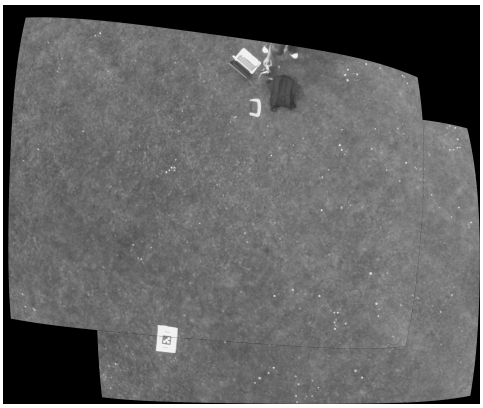
Figure 7.15: Example traversal images

as shown in Figure(s) 7.16a and 7.16b which limited the performance of the planner. Tarmac imagery also featured small areas of non-traversability caused by petals and leaves, though not to the same extent. Although paved imagery didn't particularly suffer from the effects of leaves, the areas between slabs, huge variability in surface pattern and broken slabs did tend to cause some misclassification of unsafe areas as shown in Figure(s) 7.17a and 7.17b, which did affect the planner.

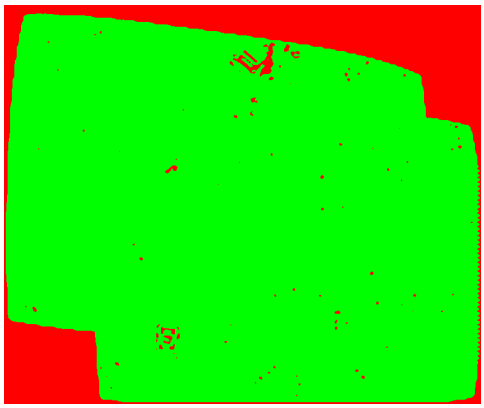
7.7.1.2 Unsafe area traversal

As well as demonstrating performance on safe paths, unsafe paths have also been trialled. Unsafe paths attempt to traverse two non-similar materials in the image e.g. grass to tarmac and should, therefore, fail to find a route. Similar to the safe area traversal, paths were selected manually with an attempt to cover different areas of the images. For each test image, three paths were attempted, with each repeated five times. Results of the unsafe area traversal image are shown in Table 7.17.

As predicted, most of the tests failed to generate a path between the two different ground materials. There was actually only a single test instance, repeated five times where a path was found. All materials appeared to perform similarly, except for the single erroneous instance using paving. Likewise with changes in height there appeared to be little discernable difference except for the single erroneous instance at seven metres.

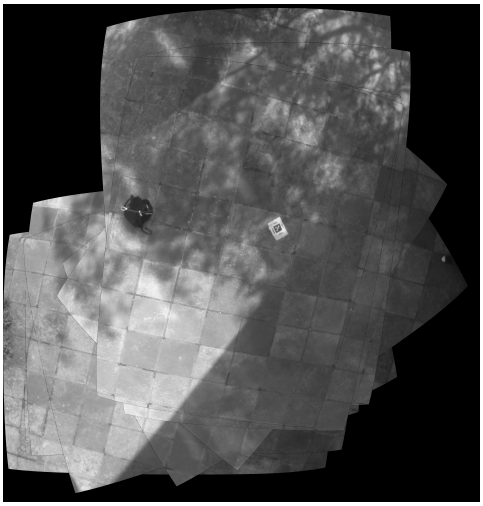


(a) Grass 5 metre flying height



(b) Binary thresholded safety image

Figure 7.16: Small obstruction example



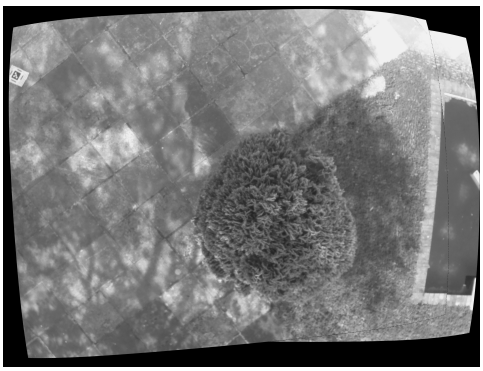
(a) Paving 5 metre flying height

(b) Paving thresholded image

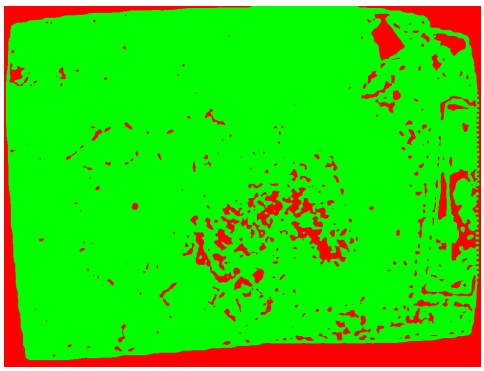
Figure 7.17: Paving example

Table 7.17: Correctly unplanned unsafe paths (%)

	Tarmac	**Grass**	**Paving**	**Average**
3m flying height	100	100	100	100
5m flying height	100	100	100	100
7m flying height	100	100	66.67	88.89
Average	100	100	88.89	



(a) Paving 7 metre flying height



(b) Paving thresholded image

Figure 7.18: Unsafe paving example

Table 7.18: Closed correctly planned safe paths (%)

	Tarmac	**Grass**	**Paving**	**Average**
3m flying height	86.67	100	0	62.22
5m flying height	100	33.33	33.33	55.56
7m flying height	33.33	26.67	66.67	42.22
Average	73.33	53.33	33.33	

The single erroneous path planned was caused by the misclassification of the goal location (a bush) as a safe area. The stitched image and binary thresholded safe/unsafe image are shown in Figure(s) 7.18a and 7.18b. Figure 7.18b, shows that the traversability similarity measure failed to identify the bush in Figure 7.18a which was then traversed in the failed test.

7.7.1.3 Morphological closing

Though the unsafe paths were successfully identified as unplannable, the safe paths suffered from the presence of small areas of non-traversability in many cases, limiting the performance of the planner. Closing (dilation followed by erosion), can be used to fill in these small areas of measured non-traversability, given that in many cases they are actually traversable. In order to evaluate this, the tests in 7.7.1.1 and 7.7.1.2 were repeated with the addition of a single closing (dilation followed by erosion) step. The goal of the closing step is to remove insignificant noise from our traversable area maps whilst at the same time keeping dangerous areas identifiable as unsafe. Results of safe path testing are shown in Table(s) 7.18 and 7.19, whereas unsafe planning is shown in Table 7.20.

After the addition of the morphological closing step, there was a clear increase in the percentage of safe paths detected overall, though path planning at seven metre heights on Tarmac decreased. The small decrease in paths at seven metres, equal to a single instance

Table 7.19: Closed correctly planned path distance ratio (actual planned distance / optimal distance)

	Tarmac	**Grass**	**Paving**	**Average**
3m flying height	1.00	1.02	N/A	1.01
5m flying height	1.05	1.00	1.00	1.02
7m flying height	1.26	1.52	1.54	1.44
Average	1.10	1.18	1.27	

Table 7.20: Closed correctly unplanned unsafe paths (%)

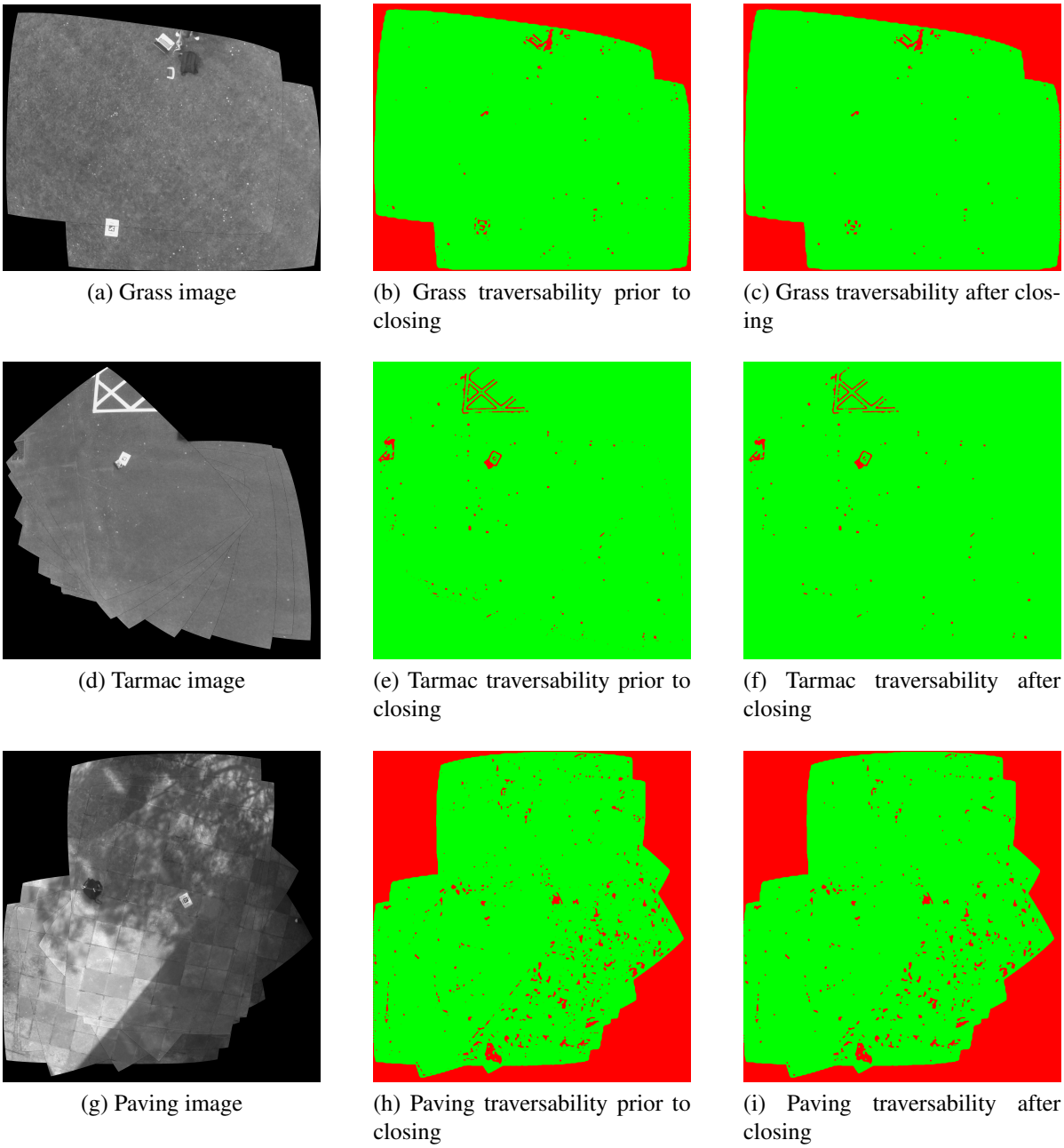
	Tarmac	**Grass**	**Paving**	**Average**
3m flying height	100	100	100	100
5m flying height	100	100	100	100
7m flying height	100	100	66.67	88.89
Average	100	100	88.89	

of a path not being planned in the second step, is perhaps due to the imposed limit on the number of tree growing iterations for the RRT, rather than being caused by the closing step. In terms of path distance ratio, again there was a clear decrease in path distance ratio, due to the straightening of convoluted paths that were initially planned to avoid small anomalous non-traversable areas. For unsafe paths, the results were exactly the same as the previous tests indicating no reduction in safety with the addition of this particular closing step. The closing step selected for this test simply sought to evaluate closing for noise reduction in traversal images, and used the default OpenCV settings with a single iteration, resulting in an improvement in performance in outdoor imagery. In order to incorporate a closing step into the full system, further evaluation of the effect of adding the step, particularly with different kernels and number of iterations would be required. Appropriate closing parameters would need to remove insignificant noise in images, whilst retaining unsafe areas or obstructions in the traversability map. Examples of the effects of closing are shown in Figure 7.19.

7.7.1.4 Discussion

This section has demonstrated the traversability and navigation system on multiple surfaces, and at multiple heights.

For the safe paths, Tarmac tended to be the most successful material for traversal and planning, given its fairly uniform surface which enabled texture similarity to be easily identified in comparison to grass and paving which tended to be less uniform, grass because it is a natural object and paving because the slabs different colours with gaps and slightly different surfaces which made ascertaining similarity difficult. In terms of height, the five metre flying height



(a) Grass image

(b) Grass traversability prior to closing

(c) Grass traversability after closing

(d) Tarmac image

(e) Tarmac traversability prior to closing

(f) Tarmac traversability after closing

(g) Paving image

(h) Paving traversability prior to closing

(i) Paving traversability after closing

Figure 7.19: Morphological closing example images, height 5m

performed best, providing enough resolution of objects to generate features whilst also being high enough to ignore certain objects such as leaves which were misclassified at lower heights. Unfortunately, the series of tests did not perform as well as expected, typically due to unexpected non-traversability caused by leaves for both the grass and tarmac environments and the varying nature of the paving in the paved environment. For the unsafe paths, most paths were unplanned as expected, unfortunately, a single set of paths were planned where the texture similarity measure failed to identify a bush as un-traversable causing a path to be planned through the object.

In its current state, the system seems to err on the side of caution, in that it tends to identify safe areas as unsafe, except for the single instance of the bush being misidentified. Unfortunately, the cautionary behaviour of the current software prevents it from planning a route in many cases due to interference from small ground objects such as leaves that though traversable are identified as unsafe, and that were not present in the previous threshold test.

Given the unexpectedly poor performance of the mapper/planner in comparison to the previous threshold testing, the insertion of a morphological closing step was investigated, with an improvement in the quantity and quality of paths planned using a single iteration. Addition of morphological closing to remove noise at lower heights should, therefore, be considered although the selection of an appropriate kernel and number of iterations would be required to maximise noise removal whilst maintaining safe identification of obstacles.

7.7.2 Stitcher evaluation

In the normal course of testing, the stitcher would have been evaluated given the slip during initialisation, then using the final real-world location versus the final calculated location after movement, but given the failure of the majority of movement tests, this wasn't possible. As an alternative, images containing the target were captured on different materials and at different heights. The captured imagery was subsequently stitched, and given the non-movement of the target, the "slip" of the stitcher was calculated over time using the distance from the start location. For each test, two hundred frames were captured and stitched, noting the slip in mm from the first reference frame at each stitch. For each height and material, the test was repeated three times.

7.7.2.1 Height

For these tests, the height was varied between three, five and seven metres above the ground at for each of the three test locations. Data is shown in Table(s) 7.21,7.22 and 7.23 corresponding to Figure(s) 7.20,7.21 and 7.22. Average texture values for each height are visualised in Figure

Table 7.21: Stitcher slip height 3 metres

Number of frames	Tarmac stitcher slip (mm)	Grass stitcher slip (mm)	Paving stitcher slip (mm)	Average stitcher slip (mm)
10	4.40	3.50	1.55	3.15
20	15.93	4.73	2.41	7.69
30	31.49	3.59	3.71	12.93
40	37.17	6.25	4.20	15.87
50	37.93	6.75	6.10	16.93
60	35.93	8.05	6.36	16.78
70	50.09	8.10	8.26	22.15
80	49.279	10.47	8.33	22.6
90	70.35	11.70	9.72	30.59
100	66.86	14.44	11.14	30.81
110	66.61	15.12	10.81	30.85
120	68.84	19.63	12.53	33.67
130	67.56	19.56	12.58	33.23
140	90.74	17.71	14.25	40.90
150	90.68	19.36	14.19	41.41
160	118.09	22.00	15.40	51.83
170	120.12	24.19	17.11	53.81
180	119.97	26.41	19.53	55.30
190	122.48	26.58	20.92	56.66
200	122.20	30.55	22.41	58.39

7.23.

At a three metre height, all of the materials increased their slip over time, likely due to additive mis-stitching over time. Grass and paving both increased at a similar, fairly level rate, probably due to a fairly constant mis-stitching per frame. Interestingly, tarmac differed in that its slip was much more erratic over time, the slip tended to plateau and rise which potentially indicates mis-identified features in certain frames causing a rise in the slip, then all being well for a certain time until another mis-stitched image is added. The cause of the difference between tarmac and the other materials is likely due to the uniformity of the tarmac, which makes feature detection more difficult and inaccurate.

At a five metre height, the slip of both tarmac and paving remained fairly constant whereas the grass slippage increased at a fairly level rate. In comparison to the lower height, the increase in grass slippage was much higher, potentially due to the lack of visible image features at the increased height causing a fairly constant level of additive slippage.

At seven metres, the grass increased in a similar way to the grass at five metres though

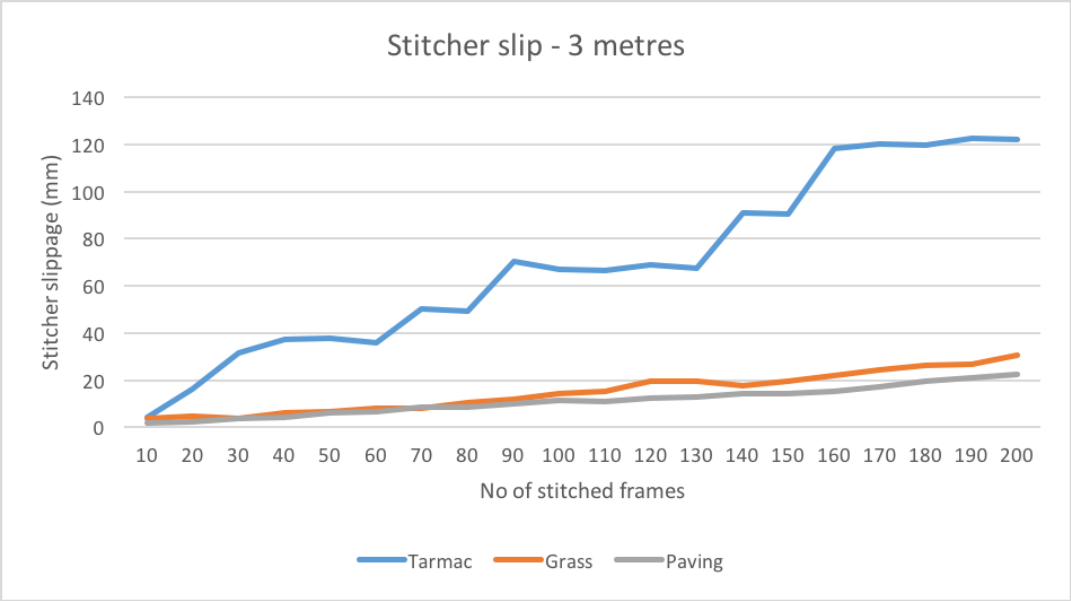


Figure 7.20: Stitcher slip height 3 metres

Table 7.22: Stitcher slip height 5 metres

Number of frames	**Tarmac stitcher slip (mm)**	**Grass stitcher slip (mm)**	**Paving stitcher slip (mm)**	**Average stitcher slip (mm)**
10	2.91	11.77	8.48	7.72
20	3.38	12.39	18.18	11.32
30	3.91	14.19	20.14	12.75
40	4.24	21.59	13.06	12.97
50	3.90	23.02	14.30	13.74
60	5.46	28.19	13.15	15.60
70	3.43	33.81	11.79	16.34
80	4.80	44.15	6.03	18.33
90	4.08	46.98	11.28	20.78
100	4.41	46.67	4.52	18.53
110	4.13	48.05	15.99	22.72
120	3.06	57.75	12.04	24.28
130	4.30	59.50	7.46	23.75
140	4.27	62.72	6.69	24.56
150	5.64	61.74	10.82	26.07
160	6.37	65.97	12.41	28.25
170	6.47	70.40	11.80	29.56
180	6.31	73.52	12.52	30.78
190	7.15	74.52	13.53	31.73
200	10.67	81.88	6.76	33.10

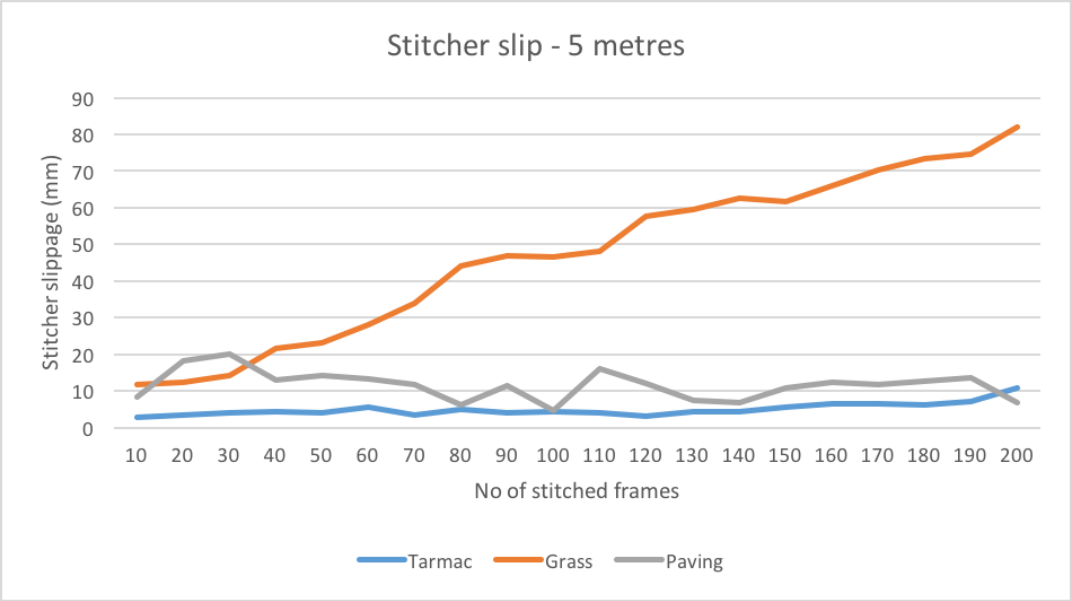


Figure 7.21: Stitcher slip height 5 metres

Table 7.23: Stitcher slip height 7 metres

Number of frames	Tarmac stitcher slip (mm)	Grass stitcher slip (mm)	Paving stitcher slip (mm)	Average stitcher slip (mm)
10	2.95	9.49	53.54	21.99
20	4.29	23.33	42.71	23.44
30	3.74	19.09	46.38	23.07
40	9.49	31.59	33.04	24.71
50	9.40	39.03	36.08	28.17
60	11.03	35.23	35.11	27.12
70	15.59	49.08	34.78	33.15
80	15.35	61.73	37.40	38.16
90	17.44	62.17	56.20	45.27
100	19.29	79.01	47.22	48.51
110	16.38	79.05	46.98	47.47
120	22.79	84.32	23.77	43.63
130	27.11	102.02	43.54	57.56
140	28.55	95.83	43.54	58.66
150	28.30	100.42	52.25	60.32
160	29.97	98.72	47.71	58.80
170	31.42	107.51	52.17	63.70
180	31.00	123.28	59.78	71.35
190	30.39	131.54	62.72	74.88
200	36.71	137.81	30.02	68.18

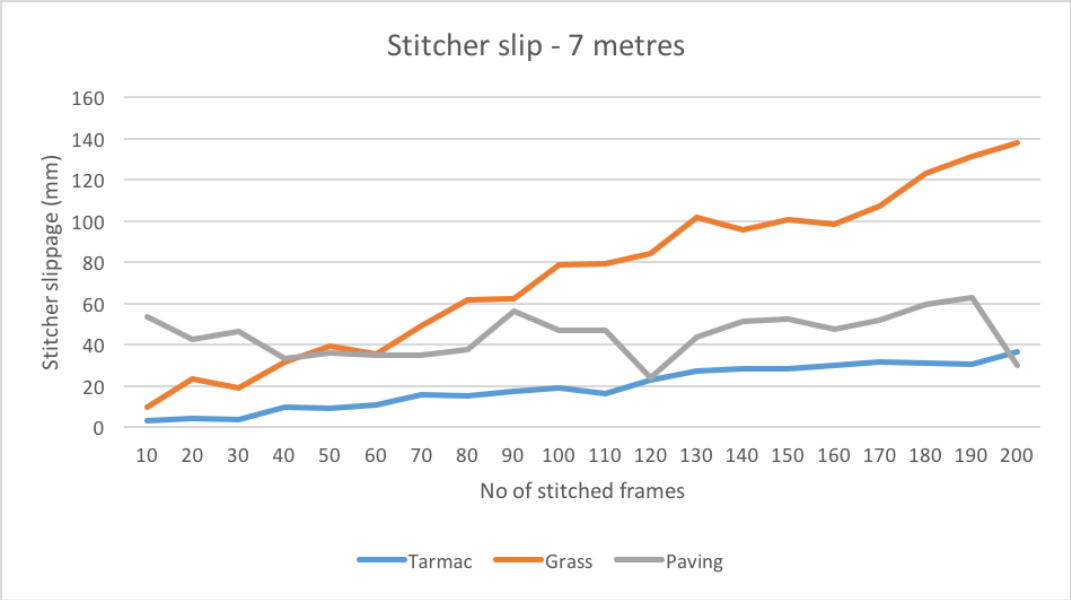


Figure 7.22: Stitcher slip height 7 metres

at a faster rate, again potentially due to a constant shortage of useful image features being identified in each frame. Paving seemed to remain reasonably constant, the many corners of the paving potentially providing a reasonable number of image features. Tarmac rose at a fairly constant rate, consistent with a small mis-stitch causing slip in every frame.

On average, as shown in Figure 7.23, the seven metre high imagery stitched more poorly than the five or three metre high images, potentially due to a lack of visible features at that height. Five and three metre images both rose over time at a reasonably similar rate, though five metre images typically had a lower slip, quite possibly due to the larger field of view meaning more overlap between images and better matches.

7.7.2.2 Material

Similar to height, the material was varied during tests for each of the heights. Data is shown in Table(s) 7.24, 7.25and7.25 corresponding to Figure(s) 7.24, 7.25 and 7.26. Average texture values for each height are shown in Figure 7.27.

Tarmac seemed to perform best at a five metre height with significantly lower slippage than both three or seven metres. At three metres, there is a plateauing of the slippage indicating a few images where there was a significant mis-stitch, potentially caused by a relatively small field of view at this height meaning that in some images there is little overlap, necessitating matching using bad quality features, and causing slip. At five metres there is little rise in slippage, potentially the larger field of view making it more likely that images overlap

Figure 7.23: Average stitcher slip at varying heights

Table 7.24: Stitcher slip tarmac

Number of frames	Flying height 3m stitcher slip (mm)	Flying height 5m stitcher slip (mm)	Flying height 7m stitcher slip (mm)	Average stitcher slip (mm)
10	4.40	2.91	2.95	3.42
20	15.93	3.38	4.29	7.86
30	31.49	3.91	3.74	13.05
40	37.17	4.24	9.49	16.97
50	37.93	3.90	9.40	17.08
60	35.93	5.46	11.03	17.47
70	50.09	3.43	15.59	23.04
80	49.279	4.80	15.35	23.14
90	70.35	4.08	17.44	30.62
100	66.86	4.41	19.29	30.19
110	66.61	4.13	16.38	29.04
120	68.84	3.06	22.79	31.56
130	67.56	4.30	27.11	32.99
140	90.74	4.27	28.55	41.19
150	90.68	5.64	28.30	41.54
160	118.09	6.37	29.97	51.47
170	120.12	6.47	31.42	52.67
180	119.97	6.31	31.00	52.42
190	122.48	7.15	30.39	53.34
200	122.20	10.67	36.71	56.53

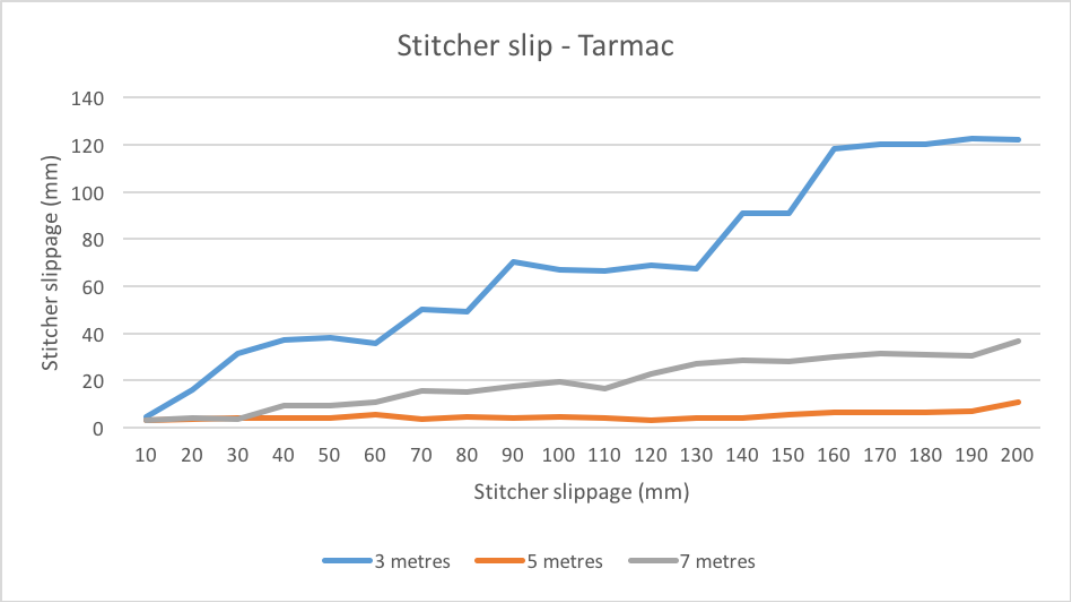


Figure 7.24: Stitcher slip tarmac

significantly but at a low enough height to identify features. At seven metres, the increased field of view is likely to mean more overlapping features between images though because of the relatively uniform nature of the material, the quality of these features is likely lower due to the increased height, offsetting the increase in field of view and leading to the rise in slip over time.

Interestingly, the stitcher slip for grass increased at a fairly even rate for all materials (indicating small additive slip over time), as well as increasing with height. The increase in slip with height is potentially due to the fine nature of the blades of grass being trickier to identify at height, despite an increase in field of view.

The slip for paving was erratic. At three metres, the additive slip increased fairly constantly. At five metres, the slip became erratic, potentially attributable to mis-matching of features between frames, the square nature of the paving and uniform pattern potentially causing, misidentification of similar features based on corners. At seven metres, the slippage was again fairly erratic, the slip distance potentially being increased by the larger field of view and subsequent mis-matching of features over a wider area.

On average, tarmac and grass both rose fairly uniformly, likely due to a constant amount of misalignment between frames. Paving (cobble), however, varied erratically, possibly due to the uniform highly-cornered nature of the material with resultant highly-similar features.

Table 7.25: Stitcher slip grass

Number of frames	Flying height 3m stitcher slip (mm)	Flying height 5m stitcher slip (mm)	Flying height 7m stitcher slip (mm)	Average stitcher slip (mm)
10	3.50	11.77	9.49	8.26
20	4.73	12.39	23.33	13.48
30	3.59	14.19	19.09	12.29
40	6.25	21.59	31.59	19.81
50	6.75	23.02	39.03	22.93
60	8.05	28.19	35.23	23.82
70	8.10	33.81	49.08	30.33
80	10.47	44.15	61.73	38.78
90	11.70	46.98	62.17	40.28
100	14.44	46.67	79.01	46.71
110	15.12	48.05	79.05	47.41
120	19.63	57.75	84.32	53.90
130	19.56	59.50	102.02	60.36
140	17.71	62.72	95.83	58.75
150	19.36	61.74	100.42	60.50
160	22.00	65.97	98.72	62.23
170	24.19	70.40	107.51	67.37
180	26.41	73.52	123.28	74.40
190	26.58	74.52	131.54	77.55
200	30.55	81.88	137.81	83.42

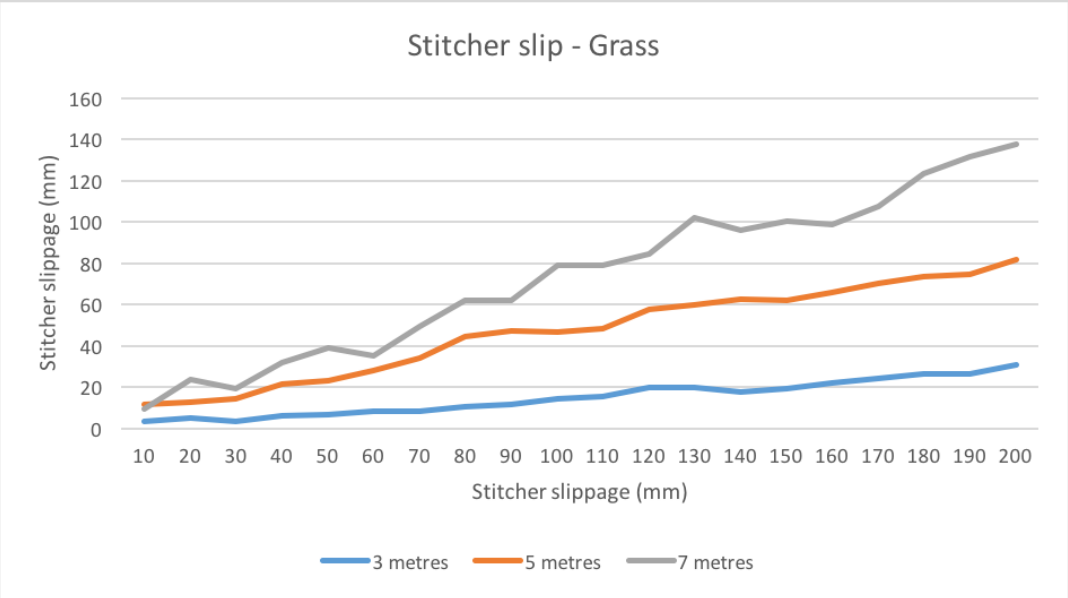


Figure 7.25: Stitcher slip grass

Table 7.26: Stitcher slip paving

Number of frames	Flying height 3m stitcher slip (mm)	Flying height 5m stitcher slip (mm)	Flying height 7m stitcher slip (mm)	Average stitcher slip (mm)
10	1.55	8.48	53.54	21.19
20	2.41	18.18	42.71	21.10
30	3.71	20.14	46.38	23.41
40	4.20	13.06	33.04	16.77
50	6.10	14.30	36.08	18.83
60	6.36	13.15	35.11	18.21
70	8.26	11.79	34.78	18.28
80	8.33	6.03	37.40	17.25
90	9.72	11.28	56.20	25.73
100	11.14	4.52	47.22	20.96
110	10.81	15.99	46.98	24.59
120	12.53	12.04	23.77	16.12
130	12.58	7.46	43.54	21.19
140	14.25	6.69	43.54	24.18
150	14.19	10.82	52.25	25.75
160	15.40	12.41	47.71	25.17
170	17.11	11.80	52.17	27.03
180	19.53	12.52	59.78	30.61
190	20.92	13.53	62.72	32.39
200	22.41	6.76	30.02	19.73

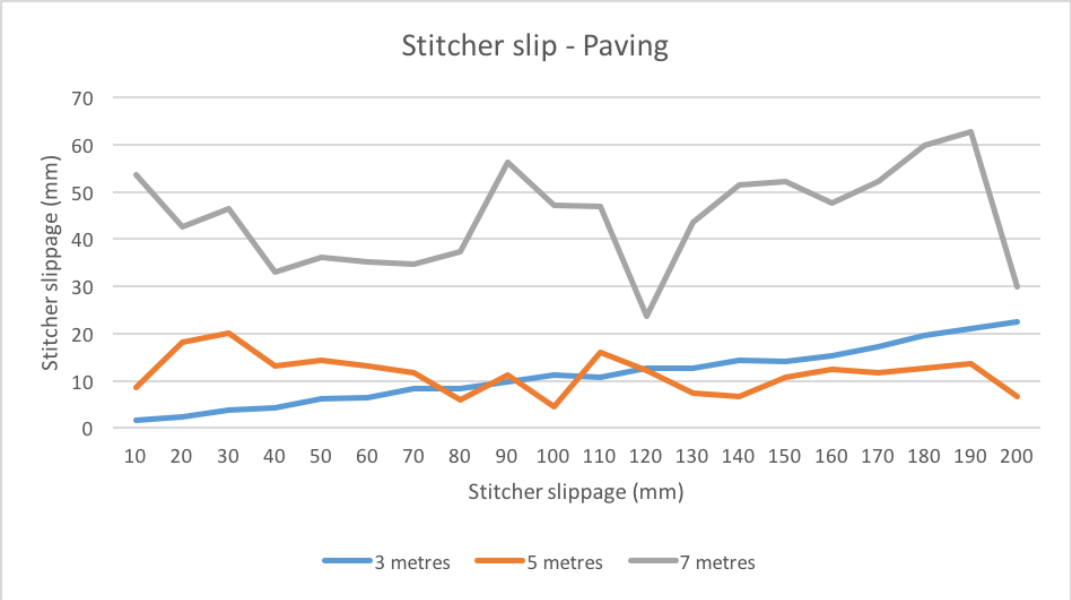


Figure 7.26: Stitcher slip paving

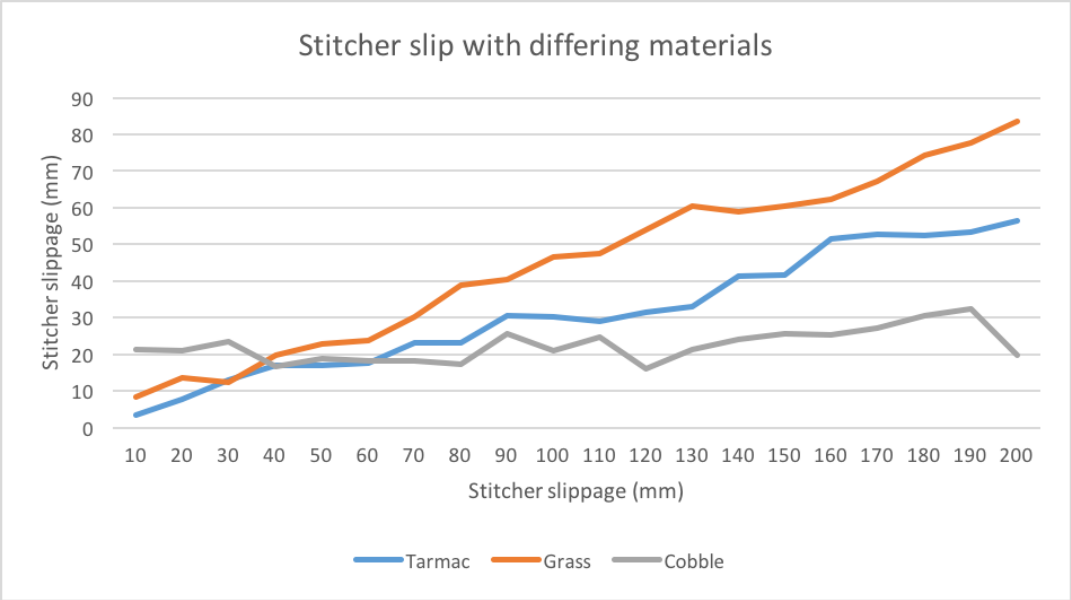


Figure 7.27: Average material slip

7.7.2.3 Discussion

This section demonstrated the amount of stitcher slip on multiple surfaces, and at multiple heights.

Height was a significant factor in stitcher slip, generally speaking, an increase in height caused a larger field of view, with more opportunity for overlapping matches between frames, however, in practice this also reduced the quality of the matches due to the decreased resolution, resulting in a tradeoff. Typically, five metre high images performed best all round with an increased field of view than the three metre imagery, whilst still retaining the ability to generate good quality matches at that height.

Performance over materials differed, tarmac and grass rose fairly uniformly, though tarmac was less sensitive to changes in height. Paving was more erratic, possibly due to the number of very similar matches (due to its ordered and cornered patterning) being mismatched, particularly as height and field of view increased.

In practical terms, the stitcher performed well over multiple materials and at multiple heights. The maximum slip was approximately 130mm which is likely acceptable, though this could be reduced by tailoring the flying height to the particular material. Objectively speaking, the stitcher slip in these outdoor tests appeared to be lower than the indoor testing, probably due to the increased number of image features in natural environments.

7.7.3 Discussion

The evaluation of the traversability planner and slip in stitcher suggest that the software component of the system should function (with minor modifications) in an outdoor environment.

Though the mapper/planner failed to initially identify safe routes in the imagery, it successfully failed to identify routes in unsafe imagery which is arguably more crucial in terms of safety (erring on the side of caution). The mapping and planning system performed unexpectedly poorly in comparison to the threshold testing in Chapter 6, likely due to the decreased height allowing for better identification of small areas of perceived non-traversability, as well as seasonal variations in the amount of these objects present between tests. For the unsafe paths, a single safe path was erroneously identified, passing through a bush, the semi-visible nature of which may have contributed to its misclassification though in terms of safety, this would have been unlikely to cause any damage to our ground vehicle. After the presence of small areas of non-traversbility in the images, Morphological closing was investigated as a method of noise removal, results were promising with a decrease in the amount of unsuccessfully planned safe plans and increase towards optimality of the path lengths. In order to implement closing in the final system, selection of appropriate parameters would have been necessary which was beyond the scope of these tests.

Stitcher slip was measured on three different surfaces and at different heights. Height was a significant factor in stitcher slip, leading to a tradeoff between the field of view and quality of matches. Overall, the slip seemed to be acceptable, and less than the slip in the indoor tests, possibly due to the increase in detected image features.

Overall, none of the materials presented any insurmountable challenges to the system, though some modifications may be required to reduce noise. A five metre flying height seemed to be optimal both for the stitcher and planner, and for all materials. Whilst some materials performed better at other heights, the five metre flying height provides a consistently good balance of feature detection and field of view on the three chosen materials.

7.8 Closing the loop discussion

The initial indoor series of tests were designed to evaluate all functionality of the closed loop system in terms of speed, optimality and robustness. A series of sixty main runs were completed though without the powered movement system which would be subsequently be evaluated in the outdoor testing. Capture, detection and stitching appeared to function correctly though there was slippage of the stitcher over the course of execution likely due to a lack of features of the ground surface and low flying height during execution. After transmission, map generation and planning with single and multi-point paths functioned on the indoor surfaces,

avoiding collision using the RRT algorithm, which was shown to be close to A* optimal in terms of path length. In some cases, the planned path distance was actually less than A*, due to the ability of the RRT to plan at any angle. Robot control was non-optimal because of the latency introduced by the low-powered nature of the system and the complex calculations in real time, causing a large difference between planned and executed path. Despite the latency causing overturning and overshooting of the path, no obstacles were hit during the indoor testing. The system as a whole is definitely not fast, with the execution of paths taking multiple minutes to complete, the root cause being latency requiring a slow movement speed. In practice, the slow speed of the robot would perhaps be more suitable to long distance paths with long sections of forward motion, rather than costly rotations. Although the planned paths are acceptably optimal, the execution reduces the optimality but does manage to complete.

After the indoor testing, some outdoor testing was proposed. Tests were run at longer distances at a higher flying height with the addition of the powered motor system and on two alternative surfaces. Unfortunately, the majority of the powered testing failed to complete, due mostly to the lack of sufficient power of the Helikite in the outdoor environment, and some failure to generate paths. The relative lack of power of the Helikite motor system, combined with unexpected air movement generally lead to the Helikite vehicle being unable to keep up with the ground vehicle, and when it did the erratic flight pattern tended to cause non-consecutive image detections, leading the stitcher to fail. In other cases, the system failed to generate paths despite keeping the target in view, after subsequent alternative tests, this may be due to the presence of small in-scene objects being identified as non-traversable.

Subsequently, two series of alternative outdoor tests were proposed which sought to quantify the suitability of the two ground materials at different heights. For the ground traversal tests, traversal of safe areas didn't perform as expected due to the presence of small areas of non-traversability in the images, whereas for safe planning, the system largely performed as expected though there was an erroneous path planned with a bush was not recognised as a dissimilar material. Overall, tarmac did seem to perform better for ground traversal, likely due to its relatively uniform surface pattern and colour in comparison to grass and paving. For height, flights at five metres performed best for both safe and unsafe areas, probably due to a pixel size that was able to quantify features effectively, whilst at the same time being able to ignore small anomalies on the same surface material. In order to solve the mis-identification of unsafe routes on similar materials, an erosion/dilation step could be used at the map updating stage in order to remove the noise. For stitcher slip, there was a tradeoff between quality and quantity of matches at height, with a five metre flying height providing the best results on average. In terms of materials, tarmac was most consistent, with grass and paving performing poorly at higher altitudes.

Chapter 8

Conclusion and suggestions for future work

This chapter compares the research questions and hypotheses to the outputs of the project. Issues and gaps in knowledge are also discussed with a view to identifying improvements and potential future research.

8.1 Completed work

Over the course of this project, a wide breadth of work has been completed in the areas of cooperative air-ground robotics, platform control, real-time image capture, texture conversion and path planning under the auspices of implementing a closed loop system for navigating unknown environments using texture similarity as a measure of safety. A low powered method of target detection was implemented and expanded based on prior work, providing location, scale and heading. A Helikite platform was selected based on its flight properties and novelty and a powered following system was implemented using a fixed tether suitable for vehicle following in low wind conditions. Continuous real-time stitching using ORB features on low powered hardware provides stitched images and locational updates. Subsequent to the stitching, a method of texture conversion using GLCMs with a combination of correlation, contrast and entropy was used to reduce images prior transmission to the GroundPi unit. A system of navigation using the textural images from the FlyPi was considered and evaluated using the assumption that start areas are safe. Assuming that the start area is safe, a measure of texture similarity is proposed based on the difference from the start texture. Using a map generated from this texture similarity map, a closed loop system of planning was implemented using a binary threshold for safety. The binary threshold T was set after evaluating traversability

using test images from the system. Using the plans, paths were executed with changes to maps leading to full or partial replanning during execution. The whole system was evaluated indoors, with success in navigating small distances, it was then trialled over longer distances and different surfaces outdoors.

8.2 Research questions and hypotheses

In the introduction, a series of research questions and hypotheses were proposed breaking the task into distinct pieces.

- **How can ground vehicles be detected by aerial vehicles?** Various options were proposed, tracking of the vehicle itself using techniques such as frame to frame matching were discounted due to the training requirements as well as uncertain motion and a non-fixed viewpoint. The chosen solution, a target based detector provided real-time locational updates on low powered hardware.

- **How can a ground-based robotic vehicle be kept in view by an aerial vehicle?** As the Helikite is tethered to the ground vehicle, the motion of the aerial robot is somewhat constrained. The proposed system implemented following using an altered proportional system that works in little or no wind. In conditions where there are winds, it is unlikely that the vehicle can be followed completely due to the power requirements needed to counter the effect of these strong winds.

- **How can positional information be captured with movement in both ground and aerial vehicles?** ORB feature points were used to stitch consecutive images in order to build a continuously stitched map from start to goal and provide location relative to the start point.

- **Which measures for quantizing pixel properties allow effective real-time segmentation of aerial images?** Colour was disregarded due to its poor performance particularly with shadows and changing light levels in natural environments. Texture was proposed as a measure for classification of similar areas using speeded-up GLCMs, and a combination of contrast, correlation and entropy.

- **How can paths be categorized as safe or unsafe using pixel values?** In order to save time during execution and to avoid the need for training, a binary system of safety based on a thresholded similarity measure from the start point was proposed.

- **Which path planning methods are suitable for real-time use on low powered hardware?** To function in real time, graph-based planners such as A* and D* were discounted, the chosen solution, based on the RRT algorithm was chosen. Although the RRT is not optimal, its speed allowed for replanning during execution.
- **Can a cooperative air and ground system be implemented that conforms to the above principles?** A system was implemented and evaluated using the overhead view of a ground vehicle from a Helikite. The computational hardware comprised of low powered and low cost equipment that successfully traversed paths in real time. Unfortunately, in outdoor conditions, the test platform was underpowered but it did succeed in providing a proof of concept.

The introduction also proposed a series of hypotheses, based on these research questions.

- Ground-based vehicles can be detected and kept in view by an aerial platform.
- Real-time segmentation into safe and unsafe routes is possible using aerial platforms.
- Navigation of these routes could be achieved using a cooperative system formed of low powered hardware without training.

The three hypotheses were confirmed, albeit with caveats. The ground vehicle was successfully detected and can be followed though not consistently in windy outdoor environments. The current test platform is underpowered and could perhaps benefit from improvements to power, robustness and speed. Real-time segmentation was achieved using the textural measures of contrast, correlation and entropy and a derivative of the GLCM. Navigation was achieved using a measure of texture similarity (though it suffered from interference from noise) and planned using a binary threshold in a cooperative manner. The proposed system succeeded in allowing the ground vehicle to reach its goal although with a lag between capture and updating, which was unhelpful in outdoor environments where erratic Helikite movement caused by wind resulted in large differences between consecutive frames.

8.3 Issues

Though the majority of the project was completed, the performance of the system itself could be improved. The detection of targets could be further improved, which would likely improve the following of the ground vehicle. The outdoor testing did expose some issues with the identification of small areas of non-traversability caused by leaves and other similar objects in view that were not previously identified due to the particular test area used, season and

flying height. Subsequent testing did evaluate the performance of a morphological closing step on these areas of non-traversability with performance increases, though implementing this would require further evaluation to ascertain appropriate parameters, and may add to the latency already present.

By far the two biggest problems with the current implementation are latency and a lack of motor power with the current test platform. The latency present between the capture of imagery and updating of the map and the resultant delay causes slow updating of the map to avoid obstacles, current location and heading, subsequent addition of the Helikite movement code further added to this latency in outdoor testing. The latency results in the ground vehicle coming close to obstacles or overshooting the goal due to the delay between reaching the goal and detection of reaching the goal state. The lack of motor power in the outdoor testing combined with the increased air movement caused by towing, and longer execution times causes in many cases the ground vehicle to be out of frame. Attempts to remedy the lack of power caused current overdrawing, effectively limiting the maximum wind speed. Underpowering of the motor ultimately resulted in the failure of many of the outdoor test runs due to the ground vehicle being out of view.

8.4 Contribution

The project has covered a wide range of work, and made original contributions in several areas. Low powered detection of targets was evaluated and a method based on prior work by Neal Snooke, which was improved upon to generate locational information in real time. The Helikite platform which had previously been little has been evaluated, its flight characteristics under power were also evaluated and a system proposed for the basic following of a towed target vehicle. Real-time stitching and texture quantification of aerial images using GLCMs was implemented on lower powered hardware. Similarity of area based on the difference from start texture was proposed as a measure of describing safety which has the advantage of being untrained except for a threshold value. All of the work is implemented in a closed loop system using minimal information and external inputs

8.5 Suggestions and ideas for future work

Suggestions for future work would be to resolve some of the problems identified during the project. Unfortunately, despite pre-testing of the constituent parts of the system (movement and capture, stitching, and mapping), the success rate of the outdoor testing was poor because of the lack of sufficient motor power, and latency, which combined with the swift Helikite

movement causes issues with keeping the ground vehicle in frame, and stitching the imagery to provide locational updates.

Updates to the software would be to implement morphological closing with appropriate parameters in order to remove noise from the traversability map, and work to improve the execution speed of the more time-expensive calculations (particularly stitching and transmission).

For hardware, any further work would require the building of a new, improved FlyPi unit due to the current unit being underpowered (though it provided the proof of concept required), for use with the current or a larger Helikite. An improved FlyPi should include an improved single board computer unit such as the Raspberry Pi 3B+ with increased processing power and faster transmission capability to remove some of the system latency. To improve the performance in wind, the motor could be replaced with a EDF (electronic ducted fan) for more efficient use of motor power, and a new tether produced capable of handling more current.

In terms of testing, after resolving motor and latency issues, further work could evaluate multiple static or introduced obstacles, introduce moving obstacles, or vary the ground vehicle used, perhaps replacing it with a waterborne vehicle in order to evaluate the performance of the system in a vastly different environment.

References

[1] J. Choi, R. E. Curry, and G. H. Elkaim, "Continuous curvature path generation based on bezier curves for autonomous vehicles," *IAENG International Journal of Applied Mathematics*, vol. 40, no. 2, pp. 91–101, 2010.

[2] C. Sprunk, "Planning motion trajectories for mobile robots using splines," tech. rep., Uni Freiburg, 2008.

[3] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, Dec. 1959.

[4] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, pp. 100–107, July 1968.

[5] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pp. 3310–3317 vol.4, IEEE, May 1994.

[6] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *Robotics, IEEE Transactions on*, vol. 21, pp. 354–363, June 2005.

[7] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong Planning A*," *Artificial Intelligence*, vol. 155, pp. 93–146, May 2004.

[8] M. Przybylski and B. Putz, "D* extra lite: A dynamic a* with search-tree cutting and frontier-gap repairing," *International Journal of Applied Mathematics and Computer Science*, vol. 27, Jan. 2017.

[9] M. Likhachev, G. Gordon, and S. Thrun, "Ara*: Anytime a* with provable bounds on sub-optimality," *NIPS (pp. 767-774).*, 2003.

[10] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic a*: An anytime, replanning algorithm," in *In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS*, 2005.

[11] A. Botea, M. Müller, and J. Schaeffer, "Near optimal hierarchical path-finding," *Journal of game development*, vol. 1, no. 1, pp. 7–28, 2004.

[12] A. Nash, K. Daniel, S. Koenig, and A. Felner, "Thetaˆ*: Any-angle path planning on grids," in *AAAI*, pp. 1177–1183, 2007.

[13] P. Yap, N. Burch, R. C. Holte, and J. Schaeffer, "Block a*: Database-driven search with applications in any-angle path-planning.," in *AAAI conference on artificial intelligence*, 2011.

[14] A. Nash, S. Koenig, and M. Likhachev, "Incremental phi*: Incremental any-angle path planning on grids," 2009.

[15] D. Ferguson and A. Stentz, "The field d* algorithm for improved path planning and replanning in uniform and non-uniform cost environments," tech. rep., Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2005.

[16] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, pp. 566–580, Aug. 1996.

[17] S. M. Lavalle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," tech. rep., Iowa State University, 1998.

[18] S. M. LaValle and J. J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, vol. 20, pp. 378–400, May 2001.

[19] S. M. LaValle and J. J. Kuffner, "Rapidly-Exploring Random Trees: Progress and Prospects," *Workshop on the algorithmic foundations of robotics*, 2000.

[20] R. Geraerts and M. H. Overmars, "A comparative study of probabilistic roadmap planners," in *Algorithmic Foundations of Robotics V*, pp. 43–57, Springer, 2004.

[21] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA. IEEE International Conference on*, vol. 2, pp. 995–1001 vol.2, IEEE, 2000.

[22] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 3, pp. 2383–2388 vol.3, IEEE, 2002.

[23] N. Seegmiller, J. Gassaway, E. Johnson, and J. Towler, "The Maverick planner: An efficient hierarchical planner for autonomous vehicles in unstructured environments," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2018–2023, IEEE, Sept. 2017.

[24] A. Ettlin and H. Bleuler, "Randomised Rough-Terrain Robot Motion Planning," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 5798–5803, IEEE, Oct. 2006.

[25] L. Jaillet, J. Cortés, and T. Siméon, "Transition-based RRT for path planning in continuous cost spaces," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 2145–2150, IEEE, 2008.

[26] S. Karaman and E. Frazzoli, "Incremental Sampling-based Algorithms for Optimal Motion Planning," in *Robotics: Science and Systems*, May 2010.

[27] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime Motion Planning using the RRT*," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1478–1483, IEEE, May 2011.

[28] E. Shan, B. Dai, J. Song, and Z. Sun, "A Dynamic RRT Path Planning Algorithm Based on B-Spline," in *Computational Intelligence and Design, 2009. ISCID. Second International Symposium on*, vol. 2, pp. 25–29, IEEE, Dec. 2009.

[29] D. Drake, S. Koziol, and E. Chabot, "Mobile Robot Path Planning With a Moving Goal," *IEEE Access*, vol. 6, pp. 12800–12814, 2018.

[30] C. Phan and H. H. Liu, "A cooperative uav/ugv platform for wildfire detection and fighting," in *System Simulation and Scientific Computing, 2008. ICSC 2008. Asia Simulation Conference-7th International Conference on*, pp. 494–498, IEEE, 2008.

[31] R. Rao, V. Kumar, and C. Taylor, "Visual servoing of a ugv from a uav using differential flatness," in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 1, pp. 743–748, IEEE, 2003.

[32] M. Garzón, J. Valente, D. Zapata, R. Chil, and A. Barrientos, "Towards a ground navigation system based in visual feedback provided by a mini uav," in *Proceedings of*

the IEEE Intelligent Vehicles Symposium Workshops, Alcal de Henares, Spain, vol. 37, 2012.

[33] J. H. Kim, J.-W. Kwon, and J. Seo, "Multi-uav-based stereo vision system without gps for ground obstacle mapping to assist path planning of ugv," *Electronics Letters*, vol. 50, no. 20, pp. 1431–1432, 2014.

[34] T. Stentz, A. Kelly, H. Herman, P. Rander, O. Amidi, and R. Mandelbaum, "Integrated air/ground vehicle system for semi-autonomous off-road navigation," *Robotics Institute*, p. 18, 2002.

[35] S. Ulun and M. Unel, "Coordinated motion of ugvs and a uav," in *Industrial Electronics Society, IECON 2013-39th Annual Conference of the IEEE*, pp. 4079–4084, IEEE, 2013.

[36] Z. Ziaei, R. Oftadeh, and J. Mattila, "Vision-based path coordination for multiple mobile robots with four steering wheels using an overhead camera," in *Advanced Intelligent Mechatronics (AIM), 2015 IEEE International Conference on*, pp. 261–268, IEEE, 2015.

[37] C. Papachristos and A. Tzes, "The power-tethered UAV-UGV team: A collaborative strategy for navigation in partially-mapped environments," in *Control and Automation (MED), 2014 22nd Mediterranean Conference of*, pp. 1153–1158, IEEE, June 2014.

[38] Z. Ziaei, R. Oftadeh, and J. Mattila, "Global path planning with obstacle avoidance for omnidirectional mobile robot using overhead camera," in *Mechatronics and Automation (ICMA), 2014 IEEE International Conference on*, pp. 697–704, IEEE, 2014.

[39] F. Guérin, F. Guinand, J.-F. Brethé, H. Pelvillain, *et al.*, "Uav-ugv cooperation for objects transportation in an industrial area," in *Industrial Technology (ICIT), 2015 IEEE International Conference on*, pp. 547–552, IEEE, 2015.

[40] E. Z. MacArthur, D. MacArthur, and C. Crane, "Use of cooperative unmanned air and ground vehicles for detection and disposal of mines," in *Optics East 2005*, International Society for Optics and Photonics, 2005.

[41] E. Mueggler, M. Faessler, F. Fontana, and D. Scaramuzza, "Aerial-guided navigation of a ground robot among movable obstacles," in *Safety, Security, and Rescue Robotics (SSRR), 2014 IEEE International Symposium on*, pp. 1–8, IEEE, 2014.

[42] S. Y. Choi, B. H. Choi, S. Y. Jeong, B. W. Gu, S. J. Yoo, and C. T. Rim, "Tethered aerial robots using contactless power systems for extended mission time and range," in *Energy Conversion Congress and Exposition (ECCE), 2014 IEEE*, pp. 912–916, IEEE, 2014.

[43] C. Luo, A. P. Espinosa, A. De Gloria, and R. Sgherri, "Air-ground multi-agent robot team coordination," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 6588–6591, IEEE, 2011.

[44] N. Michael, S. Shen, K. Mohta, Y. Mulgaonkar, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, *et al.*, "Collaborative mapping of an earthquake-damaged building via ground and aerial robots," *Journal of Field Robotics*, vol. 29, no. 5, pp. 832–841, 2012.

[45] B. Grocholsky, J. Keller, V. Kumar, and G. Pappas, "Cooperative air and ground surveillance," *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 16–25, 2006.

[46] M. Langerwisch, T. Wittmann, S. Thamke, T. Remmersmann, A. Tiderko, and B. Wagner, "Heterogeneous teams of unmanned ground and aerial robots for reconnaissance and surveillance-a field experiment," in *Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on*, pp. 1–6, IEEE, 2013.

[47] H. Tanner and D. Christodoulakis, "Cooperation between aerial and ground vehicle groups for reconnaissance missions," in *Decision and Control, 2006 45th IEEE Conference on*, pp. 5918–5923, IEEE, 2006.

[48] L. Chaimowicz and V. Kumar, "Aerial shepherds: Coordination among uavs and swarms of robots," in *Distributed Autonomous Robotic Systems 6*, pp. 243–252, Springer, 2007.

[49] W. Li, T. Zhang, and K. Kühnlenz, "A vision-guided autonomous quadrotor in an air-ground multi-robot system," in *Robotics and automation (ICRA), 2011 IEEE international conference on*, pp. 2980–2985, IEEE, 2011.

[50] J.-K. Lee, H. Jung, H. Hu, and D. H. Kim, "Collaborative control of uav/ugv," in *Ubiquitous Robots and Ambient Intelligence (URAI), 2014 11th International Conference on*, pp. 641–645, IEEE, 2014.

[51] A. Elfes, M. Bergerman, J. R. H. Carvalho, E. C. de Paiva, J. Ramaos, and S. S. Bueno, "Air-ground robotic ensembles for cooperative applications: Concepts and preliminary

results," in *Proceedings of the International Conference on Field and Service Robotics*, pp. 75–80, 1999.

[52] J. Yuh, "Design and control of autonomous underwater robots: A survey," *Autonomous Robots*, vol. 8, no. 1, pp. 7–24, 2000.

[53] J. C. Murray, M. J. Neal, and F. Labrosse, "Development and Deployment of an Intelligent Kite Aerial Photography Platform (iKAPP) for Site Surveying and Image Acquisition," *J. Field Robotics*, vol. 30, pp. 288–307, Mar. 2013.

[54] J. Valente, A. Barrientos, A. Martinez, and C. Fiederling, "Field tests with an aerial-ground convoy system for collaborative tasks," in *Proceedings of 8th Workshop de RoboCity2030-II: Robots Exteriores, Madrid, Spain*, pp. 233–248, 2010.

[55] E. Garone, A. Gasparri, R. Naldi, and M. Nicotra, "A Team of Cooperative Ground/Aerial Robots," tech. rep., University of Michigan, 2013.

[56] N. Michael, J. Fink, and V. Kumar, "Controlling a team of ground robots via an aerial robot," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp. 965–970, IEEE, 2007.

[57] O. De Silva, G. K. Mann, and R. G. Gosine, "Development of a relative localization scheme for ground-aerial multi-robot systems," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 870–875, IEEE, 2012.

[58] M. Garzón, J. Valente, D. Zapata, and A. Barrientos, "An aerial-ground robotic system for navigation and obstacle mapping in large outdoor areas," *Sensors*, vol. 13, no. 1, pp. 1247–1267, 2013.

[59] L. Zikou, C. Papachristos, and A. Tzes, "The Power-over-Tether system for powering small UAVs: Tethering-line tension control synthesis," in *Control and Automation (MED), 2015 23th Mediterranean Conference on*, pp. 681–687, IEEE, June 2015.

[60] L. Carlucci, "A formal system for texture languages," *Pattern Recognition*, vol. 4, no. 1, pp. 53–72, 1972.

[61] S. W. Zucker, "Toward a model of texture," *Computer Graphics and Image Processing*, vol. 5, no. 2, pp. 190–202, 1976.

[62] R. M. Haralick, K. Shanmugam, and Others, "Textural features for image classification," *IEEE Transactions on systems, man, and cybernetics*, no. 6, pp. 610–621, 1973.

[63] T. Ojala, M. Pietikäinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions," *Pattern recognition*, vol. 29, no. 1, pp. 51–59, 1996.

[64] K. I. Laws, "Textured image segmentation," tech. rep., University of Southern California Los Angeles, 1980.

[65] W. Navas and R. V. Espinosa, "Analysis of texture using the fractal model," 1997.

[66] P. Shanmugavadivu and V. Sivakumar, "Fractal dimension based texture analysis of digital images," *Procedia Engineering*, vol. 38, pp. 2981–2986, 2012.

[67] G. R. Cross and A. K. Jain, "Markov random field texture models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 1, pp. 25–39, 1983.

[68] C. Kervrann and F. Heitz, "A markov random field model-based approach to unsupervised texture segmentation using local and global spatial statistics," *IEEE transactions on image processing*, vol. 4, no. 6, pp. 856–862, 1995.

[69] S. P. Wilson and J. Zerubia, "Unsupervised segmentation of textured satellite and aerial images with bayesian methods," in *Signal Processing Conference, 2002 11th European*, pp. 1–4, IEEE, 2002.

[70] F. Zhou, J. F. Feng, and Q. Y. Shi, "Texture feature based on local fourier transform," in *Image Processing, 2001. Proceedings. 2001 International Conference on*, vol. 2, pp. 610–613, IEEE, 2001.

[71] A. C. Bovik, M. Clark, and W. S. Geisler, "Multichannel texture analysis using localized spatial filters," *IEEE transactions on pattern analysis and machine intelligence*, vol. 12, no. 1, pp. 55–73, 1990.

[72] P. Scheunders, S. Livens, G. Van de Wouwer, P. Vautrot, and D. Van Dyck, "Wavelet-based texture analysis," *International Journal of Computer Science and Information Management*.

[73] E. Karaman, U. Çinar, E. Gedik, Y. Yardımcı, and U. Halıcı, "Fourier based feature descriptors for railroad extraction from aerial images," in *Geoscience and Remote Sensing Symposium (IGARSS), 2012 IEEE International*, pp. 6013–6015, IEEE, 2012.

[74] C. Zhu and X. Yang, "Study of remote sensing image texture analysis and classification using wavelet," *International Journal of Remote Sensing*, vol. 19, no. 16, pp. 3197–3203, 1998.

[75] L. Mioulet, T. P. Breckon, A. Mouton, H. Liang, and T. Morie, "Gabor features for real-time road environment classification," in *Industrial Technology (ICIT), 2013 IEEE International Conference on*, pp. 1117–1121, IEEE, 2013.

[76] R. M. Haralick, "Statistical and structural approaches to texture," *Proceedings of the IEEE*, vol. 67, pp. 786–804, May 1979.

[77] D. A. Clausi, "An analysis of co-occurrence texture statistics as a function of grey level quantization," *Canadian Journal of remote sensing*, vol. 28, no. 1, pp. 45–62, 2002.

[78] B. Sridhar, A. Phatak, and G. B. Chatterji, "Scene segmentation of natural images using texture features and back-propagation," in *Artificial Neural Networks, 1993., Third International Conference on*, pp. 200–204, IET, 1993.

[79] F. Bianconi and A. Fernández, "Rotation invariant co-occurrence features based on digital circles and discrete fourier transform," *Pattern Recognition Letters*, vol. 48, pp. 34–41, 2014.

[80] F. R. de Siqueira, W. R. Schwartz, and H. Pedrini, "Multi-scale gray level co-occurrence matrices for texture description," *Neurocomputing*, vol. 120, pp. 336–345, 2013.

[81] F. Mirzapour and H. Ghassemian, "Using glcm and gabor filters for classification of pan images," in *Electrical Engineering (ICEE), 2013 21st Iranian Conference on*, pp. 1–6, IEEE, 2013.

[82] D. A. Clausi and M. E. Jernigan, "A fast method to determine co-occurrence texture features," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 36, no. 1, pp. 298–300, 1998.

[83] A. E. Svolos and A. Todd-Pokropek, "Time and space results of dynamic texture feature extraction in mr and ct image analysis," *IEEE transactions on information technology in biomedicine*, vol. 2, no. 2, pp. 48–54, 1998.

[84] D. A. Clausi and Y. Zhao, "Rapid extraction of image texture by co-occurrence using a hybrid data structure," *Computers & Geosciences*, vol. 28, no. 6, pp. 763–774, 2002.

[85] D. A. Clausi and Y. Zhao, "Grey level co-occurrence integrated algorithm (GLCIA): a superior computational method to rapidly determine co-occurrence probability texture features," *Computers & Geosciences*, vol. 29, no. 7, pp. 837–850, 2003.

[86] M. A. Tahir, A. Bouridane, F. Kurugollu, and A. Amira, "Accelerating the computation of glcm and haralick texture features on reconfigurable hardware," in *Image Processing, 2004. ICIP'04. 2004 International Conference on*, vol. 5, pp. 2857–2860, IEEE, 2004.

[87] J. Y. Tou, K. K. Y. Khoo, Y. H. Tay, and P. Y. Lau, "Evaluation of speed and accuracy for comparison of texture classification implementation on embedded platform," *Int. Workshop on advanced image processing*, 2009.

[88] O. Barkan, J. Weill, L. Wolf, and H. Aronowitz, "Fast high dimensional vector multiplication face recognition," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1960–1967, 2013.

[89] S. Kluckner, G. Pacher, H. Grabner, H. Bischof, and J. Bauer, "A 3d teacher for car detection in aerial images," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pp. 1–8, IEEE, 2007.

[90] G. R. Bradski, "Computer Vision Face Tracking For Use in a Perceptual User Interface," *Intel Technology Journal Q2 '98*, 1998.

[91] K. Fukunaga and L. D. Hostetler, "The estimation of the gradient of a density function, with applications in pattern recognition," *Information Theory, IEEE Transactions on*, vol. 21, pp. 32–40, Jan. 1975.

[92] E. Rosten, R. Porter, and T. Drummond, "Faster and Better: A Machine Learning Approach to Corner Detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, pp. 105–119, Jan. 2010.

[93] E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," in *Computer Vision - ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), vol. 3951 of *Lecture Notes in Computer Science*, ch. 34, pp. 430–443, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[94] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91–110, Nov. 2004.

[95] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, (Los Alamitos, CA, USA), pp. 1150–1157 vol.2, IEEE, Aug. 1999.

[96] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *Proceedings of the 7th International Joint Conference on*

Artificial Intelligence - Volume 2, IJCAI'81, (San Francisco, CA, USA), pp. 674–679, Morgan Kaufmann Publishers Inc., 1981.

[97] C. Tomasi and T. Kanade, "Shape and motion from image streams: A factorization method.," *Int. J. Comput. Vision*, 1992.

[98] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded Up Robust Features," in *Computer Vision - ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), vol. 3951 of *Lecture Notes in Computer Science*, ch. 32, pp. 404–417, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[99] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *2011 International Conference on Computer Vision*, pp. 2564–2571, Nov. 2011.

[100] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary Robust Independent Elementary Features," in *Computer Vision - ECCV 2010* (K. Daniilidis, P. Maragos, and N. Paragios, eds.), vol. 6314 of *Lecture Notes in Computer Science*, ch. 56, pp. 778–792, Berlin, Heidelberg: Springer, 2010.

[101] L. Klodt, S. Khodaverdian, and V. Willert, "Motion control for uav-ugv cooperation with visibility constraint," in *Control Applications (CCA), 2015 IEEE Conference on*, pp. 1379–1385, IEEE, 2015.

[102] S. Hood, K. Benson, P. Hamod, D. Madison, J. M. O'Kane, and I. Rekleitis, "Bird's eye view: Cooperative exploration by UGV and UAV," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 247–255, IEEE, June 2017.

[103] N. Frietsch, O. Meister, C. Schlaile, and G. Trommer, "Teaming of an ugv with a vtol-uav in urban environments," in *Position, Location and Navigation Symposium, 2008 IEEE/ION*, pp. 1278–1285, IEEE, 2008.

[104] J. McIntyre, A. Church, F. Labrosse, and Others, "Efficient image-based tracking of apparently changing moving targets," *Proceedings of Towards Autonomous Robotic Systems*, pp. 119–126, 2009.

[105] G. J. J. Verhoeven, J. Loenders, F. Vermeulen, and R. Docter, "Helikite aerial photography - a versatile means of unmanned, radio controlled, low-altitude aerial archaeology," *Archaeol. Prospect.*, vol. 16, pp. 125–138, Apr. 2009.

[106] G. Verhoeven, D. Taelman, and F. Vermeulen, "Computer vision-based orphophot mapping of complex archeological sites: The ancient quarry of pitaranha (portugal-spain)," *Archaeometry*, vol. 54, pp. 1114–1129, Dec. 2012.

[107] G. J. J. Verhoeven, "Providing an archaeological bird's-eye view an overall picture of ground-based means to execute low-altitude aerial photography (LAAP) in Archaeology," *Archaeological Prospection*, vol. 16, pp. 233–249, Oct. 2009.

[108] G. Verhoeven and J. Loenders, "Looking through black-tinted glasses - a remotely controlled infrared eye in the sky," in *From Space to Place. 2nd International Conference on Remote Sensing in Archaeology. Proceedings of the 2nd International Workshop, CNR, Rome, Italy, December 4-7, 2006*, pp. 73–79, Archaeopress, 2006.

[109] C. T. Dougherty, "Remote Sensing of Equine Bermudagrass Pastures from a Helikite." url: http://bit.ly/2nEmqK5, 2015 (Accessed February 5, 2016).

[110] M. A. Fonstad, J. T. Dietrich, B. C. Courville, J. L. Jensen, and P. E. Carbonneau, "Topographic structure from motion: a new development in photogrammetric measurement," *Earth Surf. Process. Landforms*, vol. 38, pp. 421–430, Mar. 2013.

[111] J. Young-Heon, S. Jin, K. Jae-Il, J. Kicheon, and P. Jinku, "Mapping bathymetry based on waterlines observed from low altitude Helikite remote sensing platform," *Acta Oceanologica Sinica*, vol. 34, no. 9, pp. 110–116, 2015.

[112] A. Valcarce, T. Rasheed, K. Gomez, S. Kandeepan, L. Reynaud, R. Hermenier, A. Munari, M. Mohorcic, M. Smolnikar, and I. Bucaille, "Airborne Base Stations for Emergency and Temporary Events," in *Personal Satellite Services* (R. Dhaou, A.-L. Beylot, M.-J. Montpetit, D. Lucani, and L. Mucchi, eds.), vol. 123 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 13–25, Springer International Publishing, 2013.

[113] F. B. Teixeira, T. Oliveira, M. Lopes, C. Leocadio, P. Salazar, J. Ruela, R. Campos, and M. Ricardo, "Enabling broadband internet access offshore using tethered balloons: The BLUECOM+ experience," in *OCEANS 2017 - Aberdeen*, pp. 1–7, IEEE, June 2017.

[114] C. R. Dusane, A. V. Wani, R. S. Pant, D. Chakraborty, and B. Chakravarthy, "An Elevated Balloon-Kite Hybrid platform for Surveillance," in *23rd AIAA Lighter-Than-Air Systems Technology Conference*, American Institute of Aeronautics and Astronautics, June 2017.

[115] H. Hosseini, *Automatic Selection of Image Regions for Active Fixation*. PhD thesis, Aberystwyth University, 2014.

[116] T. Ojala, M. Pietikäinen, and T. Mäenpää, "Gray Scale and Rotation Invariant Texture Classification with Local Binary Patterns," in *Computer Vision - ECCV 2000*, vol. 1842 of *Lecture Notes in Computer Science*, ch. 27, pp. 404–420, Berlin, Heidelberg: Springer Berlin Heidelberg, Apr. 2000.

[117] Bayes and Price, "An Essay towards Solving a Problem in the Doctrine of Chances. By the Late Rev. Mr. Bayes, F. R. S. Communicated by Mr. Price, in a Letter to John Canton, A. M. F. R. S.," *Philosophical Transactions*, vol. 53, pp. 370–418, Jan. 1763.

[118] T. Shan and B. Englot, "Tunable-Risk Sampling-Based Path Planning Using a Cost Hierarchy,"

[119] P. Slovic and E. Peters, "Risk Perception and Affect," *Current Directions in Psychological Science*, vol. 15, pp. 322–325, Dec. 2006.

[120] A. Zelinsky, "A mobile robot exploration algorithm," *Robotics and Automation, IEEE Transactions on*, vol. 8, pp. 707–717, Dec. 1992.

[121] L. Merckelbach, "On the probability of underwater glider loss due to collision with a ship," *Journal of Marine Science and Technology*, vol. 18, pp. 75–86, Aug. 2013.

[122] L. Murphy, S. Martin, and P. Corke, "Creating and using probabilistic costmaps from vehicle experience," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 4689–4694, IEEE, Oct. 2012.

[123] B. Pfeiffer, R. Batta, K. Klamroth, and R. Nagi, "Path Planning for UAVs in the Presence of Threat Zones Using Probabilistic Modeling," in *IEEE Transactions on automatic control*, pp. 278–283, 2007.

[124] L. Murphy and P. Newman, "Risky planning: Path planning over costmaps with a probabilistically bounded speed-accuracy tradeoff," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 3727–3732, IEEE, May 2011.